A New Metaheuristic Approach for Stabilizing the Solution Quality of Simulated Annealing and Applications

Eine neue Metaheuristik zur Stabilisierung der Lösungsqualität von simuliertem Ausglühen mit Anwendungen

For educational attainment of a doctor of computer science (Dr.-Ing.)
approved dissertation by Dipl.-Inf. Dennis Güttinger from Frankfurt (Main)
February 2013 - Darmstadt - D17

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Department of Computer Science
Knowledge Engineering

A New Metaheuristic Approach for Stabilizing the Solution Quality of Simulated Annealing and Applications
Eine neue Metaheuristik zur Stabilisierung der Lösungsqualität von simuliertem Ausglühen mit Anwendungen

Approved dissertation by Dipl.-Inf. Dennis Güttinger from Frankfurt (Main)

1st evaluation: Johannes Fürnkranz
2nd evaluation: Karsten Weihe

Day of submission: Thursday, December 13th, 2012
Day of examination: Thursday, January 31st, 2013

Darmstadt - D17

Declaration about the doctoral thesis

Herewith I affirm that I have drawn up the doctoral thesis at hand without help by any third party by using only the quoted resources and tools. All passages that were quoted from resources are marked as such. This work has not yet been presented to any examination office in equal or similar form.

Frankfurt (Main), December 13th, 2012

(D. Güttinger)

Academic Education

06/1999	High-school diploma	Albert-Einstein-Gymnasium, Maintal
10/2001-09/2006	Studies in Computer Science	Goethe University, Frankfurt/Main
09/2006	Diploma in Computer Science	Goethe University, Frankfurt/Main
10/2008-01/2013	Doctorate in Computer Science	TU Darmstadt, Darmstadt

Table of contents

1 Introduction

1.1 Problem characterization

In this work we deal with the problem of finding solutions of hard optimization problems efficiently. This question has been extensively discussed over the last decades. In the easiest case of a continuous problem where a continuously differentiable objective function is given, the optimal solution can be directly derived in closed form with standard techniques from analysis. However, only a very small minority of optimization problems that appear in practice can be expressed by an objective function with such strong analytical characteristics. Moreover, most of these problems are not even continuous but discrete. As a consequence, to find the optimal solution of a discrete problem, very often the only possibility is it to check out and compare all potential solutions. It is clear that this procedure is unefficient in general and sometimes even practically impossible to be implemented if the number of solutions is too high.
As a result, the two different groups of local and global optimization algorithms for treatment of combinatorial optimization problems have evolved. The first group is characterized by strategies that are computationally efficient but usually only compute a solution that is *locally* optimal. The other group comprises optimization heuristics, that start from any initial solution and try to approximate the global optimum by iteratively improving the initial solution. The global optimum usually cannot be determined exactly by these heuristics in finite time, but in many cases it is possible to derive good solutions very close to it efficiently.
Except for the greedy strategies, that follow a general structure, most algorithms that belong to the first group are very problem specific. On the other side, global optimization heuristics usually are designed in a way that they are applicable to as many problem types as possible. Some of them are inspired by other research fields like physics or biology, especially the most recent approaches. Another characteristic of algorithms from the second group are parameters that have to be set by the user as input. Besides, most of them require the definition of specific operators for their implementation. However, for some existing heuristics and problem types, this can be a very difficult task. Thus, from a user perspective, when designing a new heuristical approach, the number of required input parameters and operators should be as small as possible. In addition, one should also take care of the fact that the construction, comparison, and evaluation of solutions can be computationally very resource intensive, especially for complex optimization problems.

A global optimization heuristic that is frequently applied in practice is the *simulated annealing algorithm*. One reason for its popularity is that only a few operators have

to be defined, such that it can easily be adapted to many different problems. Besides, only one solution is evaluated in every iteration which is just slightly modified afterwards, such that a complete reconstruction to obtain a new solution is usually not necessary. Furthermore, only three different input parameters have to be set. This is a relatively small number in comparison to other heuristics. However, for the simulated annealing algorithm in its classic form an unfavorable choice of the input parameter values can have a strong negative impact on the solution quality as we will see in a later chapter (revise section 1.3 for a detailed overview on the structure of this work).
To avoid this problem, many different strategies have been discussed in the literature how to choose the input parameters of simulated annealing correctly. Nevertheless and to the best of our knowledge, the problem of the optimal input parameter choice is still open and there is not much hope that the answer to the question could be given conclusively, since the optimal parameter choice also strongly depends on the problem type.

1.2 Contributions

As a consequence of the parameter choice problem discussed in the final paragraph of the last section, we therefore suggest to change the way of looking at this problem by asking the following question: If it is already impossible to give a general guidance for choosing the simulated annealing parameters, how can we achieve it that this choice is at least less relevant for the quality of our final solution?
Hence, the first main contribution of this work is a new metaheuristic approach that combines a local greedy optimization strategy with a modified version of the simulated annealing algorithm. We will empirically show that for this metaheuristic the variation of the solution quality is significantly smaller and results are also better on average than for the classic simulated annealing algorithm when choosing the input parameters randomly. So finally, the decision of finding the best input parameters is not that important for our metaheuristic compared to classic simulated annealing. We will abbreviate our strategy by GWMSA ("Greedy with modified simulated annealing") in the rest of this work and demonstrate its superiority by applying it on different combinatorial optimization problems from practice. All of these, which will be described in the following, are computationally very complex in a way that total enumeration and trying out of solutions is the only possibility to determine the globally optimal result.

The first problem is from the field of disaster management and considers the task of assigning casualties to available ambulances and qualified physicians in hospitals after a major incident. When such an incident occurs, for example after a stampede at a mass event, usually a great number of casualties need medical supply at one

specific place concurrently. As transportation capacities and medical resources are only limitedly available on site, the treatment of casualties necessarily has to be prioritized with respect to their degree of injury. In today's practice, this is done by a triage procedure, where every casualty is classified into a *triage group*. These triage groups are static and do not consider the course on an injury with respect to waiting time until medication. This is, however, inappropriate for most injured people, as their state of health changes over time (usually gets worse).
As second main contribution of this work we will therefore introduce a new penalty approach that maps waiting time until medication to a penalty value depending on the type of injury a casualty suffers from. We will formulate two new optimization problems where this concept is incorporated. The first problem is a static version, where we consider the task of planning security measures in the run-up to a mass event. Secondly, we investigate an online version which deals with the situation when a mass casualty incident has just occurred, and injured people that have to be supplied medically are continuously registered at the scene in groups. We will denote the corresponding problem class by CASASSIGN.
Moreover, we show the versatility of penalty functions by transferring this approach to another optimization problem from disaster management that arises when a list of tasks has to be allocated to a given set of roles in the context of fire fighting. This problem, which we will refer to by using the abbreviation TASKALLOC in the following, is non-trivial, as it is assumed that many different roles are qualified to perform a specific task. Furthermore, the physical and psychic stress a role suffers when performing a specific task usually differs between the roles and should thus be taken into consideration when computing an optimal allocation of tasks to roles. We will show that our concept of penalty functions can usefully be employed at this point by assigning such a function to every role. In the end, the stress a role suffers is modelled by mapping total working time to a role specific penalty value.
The third problem class we will investigate is about test suite reduction, which deals with the task of testing a collection of functions within a software program. For this purpose, test cases are available, where every test case consists of a set of function calls. The goal is to select a subset of test cases, such that every function is called at least once and the total number of function calls is minimized concurrently. As another contribution of this work we will identify the distribution over function calls as a new practical requirement. When incorporating this additional goal, the global objective function of this problem is extended by a variance term. In the following, we will denote this extension of the test suite reduction problem by TESTSUITE.
As final main contribution of this work we will provide a general implementation guidance of the GWMSA strategy on an abstract level as well as concrete implementations for the three problem classes mentioned above. We will empirically show their superiority in comparison to standard techniques that are currently used in practice to compute solutions for the CASASSIGN, TESTSUITE,

and TASKALLOC problem classes.
Besides, we will consider another problem class, WEBRANK, which deals with the problem of ranking websites that match a specific search string. Search engines usually order websites with respect to their topic relevance according to some weighting criterion. More recent approaches try to collect and employ user relevance feedbacks for optimizing search engines. We will describe an implementation of our GWMSA approach for the WEBRANK problem and empirically observe that for this problem class, the results that are computed by a very simple local approximization technique can neither be improved significantly by application of GWMSA nor another global optimization heuristic that was implemented as additional standard of comparison. So, as another small contribution of this work, we could show that for the WEBRANK problem efficient local optimization algorithms are able to compute a result that is already sufficiently close to the global optimum, although the underlying complexity of WEBRANK is also very high.

1.3 Outline

The remainder of this work is structured as follows: In the next chapter we will introduce the needed optimization principles and concepts. This includes the formal definition of a combinatorial optimization problem and an introduction of the most frequently applied metaheuristics. We will then analyze their drawbacks, which will lead us to a detailed discussion of our new metaheuristic approach in chapter 3. In addition to an abstract specification, we will also demonstrate the applicability of our concepts on a well-known combinatorial optimization problem in this chapter. The main applications of our metaheuristic will be introduced in the subsequent chapters. At this, we will first deal with both variants of the casualty assignment problem in chapter 4 and focus on the topic of extended test suite reduction afterwards in chapter 5. The task allocation problem will be discussed in chapter 6. In chapter 7 we will empirically demonstrate the parameter choice problem of the classic simulated annealing algorithm mentioned above and show the superiority of our modifications. Next, the website ranking problem will be described in chapter 8 as our final main application. In all application chapters we will first formally specify the respective problem and present corresponding implementations of our GWMSA strategy afterwards. The final sections each contain empirical simulation studies of GWMSA compared to other standard techniques that are currently used in practice to compute solutions of the given problem type.
We will give a survey of all other related articles that cover any topic which is also addressed in this work in chapter 9. All main results of this work as well as propositions for some interesting research questions for prospective studies will be finally summarized in chapter 10.

2 Optimization principles

In this chapter we will first formulate a combinatorial optimization problem in abstract form. We will then introduce some metaheuristical approaches that are frequently used in practice for solution of such problem types in section 2.2 and analyze their drawbacks.

2.1 Combinatorial optimization problems

As combinatorial optimization problems are always of discrete character, the solution space of such problems is always a finite set of elements

$$\mathcal{S} := \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}.$$

Furthermore, we assume that there exists a discrete set of components,

$$C := \{c_1, c_2, \ldots, c_{|C|}\},$$

such that every solution is a collection of those components, which means $S \subset C$ for every $S \in \mathcal{S}$.
Every collection of components $S_{part} \subset C$ which can be extended to a solution $S \in \mathcal{S}$ by adding additional components $c \in C$ to S_{part} is called partial solution. Therefore, we define

$$\mathcal{S}_{part} := \{S_{part} \subset C \,|\, \exists\, S \in \mathcal{S} \,:\, S_{part} \subset S\}.$$

The above definition of partial solutions also implies that $\emptyset \in \mathcal{S}_{part}$ and $S \in \mathcal{S}_{part}$ for every $S \in \mathcal{S}$.
Finally, let

$$F \,:\, \mathcal{S} \to \mathbb{R}$$

be the objective function that assigns a value to every solution $S \in \mathcal{S}$.

Then, solving the combinatorial optimization problem at hand generally corresponds to minimizing the value of F, subject to some given side constraints that bound the solution space $\mathcal{S}$.

In this work we will consider very hard combinatorial optimization problems that belong to the complexity class of $\mathcal{NP}$-complete problems. Although the exact definition is much more sophisticated (we refer to standard textbooks like [3] for a detailed discussion), it is sufficient for us to conceive this class as the set of those problems where a total enumeration and evaluation of all potential solutions in $\mathcal{S}$

is the only possibility to determine the global optimum exactly. Hence, we have to rely on local and global optimization heuristics to compute a solution that is at least locally optimal or is close enough to the globally optimal result.

To be able to design a local greedy strategy, we additionally assume that there exists an evaluation map

$$f : \mathcal{S}_{part} \times \mathcal{P}(C) \to \mathbb{R}_+ \cup \{0\}$$

with two specific properties. These are

i. $f(S, C_{\bar{S}}) > 0$ if $S \cap C_{\bar{S}} = \emptyset$ and $S \cup C_{\bar{S}} \in \mathcal{S}_{part}$ and

ii. $f(S, C_{\bar{S}}) = 0$ otherwise,

for every $S \in \mathcal{S}_{part}$ and every $C_{\bar{S}} \subset C$. We presuppose that higher values of f correspond to better partial solutions of the present optimization problem, which implies an inversely proportional relationship between f and the global objective function F. Thus, given a partial solution $S_{part} \in \mathcal{S}_{part}$ f allows for an ordering of the possible extensions of S_{part}. Moreover, disregarding its particular characteristics, the existence of such an evaluation map is even a necessary condition for the development of any greedy strategy that is suited to compute an approximatively optimal solution of an optimization problem. The reason is that in general, greedy algorithms form a complete solution of a problem by iteratively extending a partial solution with additional components. For this purpose, every possible extension of a partial solution has to be evaluated. Greedy will then choose that extension which is most promising on the basis of the current partial solution, see [40] for details. Concerning the codomain of f, by using translation operations it is usually straightforward to transform a given evaluation map, such that the conditions (i.) and (ii.) from above are satisfied.
In subsection 3.1.1 we will show how a general greedy strategy can be designed based on an evaluation map with the above properties. Before, we will discuss approaches for global optimization.

2.2 Metaheuristics

We will first explain the general concept of a metaheuristic in the next subsection and give examples of metaheuristical strategies that are currently used in practice afterwards. We will finally discuss the drawbacks of the existing techniques and motivate the basic ideas of our new metaheuristic approach GWMSA in subsection 2.2.7.

2.2.1 General description

Metaheuristics are abstract specifications of algorithms that can be used to compute approximatively optimal solutions of combinatorial optimization problems of the type described in section 2.1. More generally, Luke emphasizes a close relationship of metaheuristics and the concept of *stochastic optimization*: "Stochastic optimization is the general class of algorithms and techniques which employ some degree of randomness to find optimal (or as optimal as possible) solutions to hard problems. Metaheuristics are the most general of these kinds of algorithms, and are applied to a very wide range of problems" ([104], p.7).
We will exclusively consider problems in this work where the globally optimal solution can only be determined by total enumeration and evaluation of the whole solution space. Hence, it is clear that it cannot be guaranteed that this optimal solution is found by a metaheuristic in finite time. In many cases it is, however, possible to determine at least a solution that is sufficiently close enough to the global optimum.

Metaheuristics usually are of iterative form in the sense that one or more initial solutions are iteratively improved according to some optimization criterion (see following subsections for concrete examples). The procedure can thus be summarized by the following abstract steps:

1. Compute one (ore more) initial solution(s).
2. Repeat until some termination condition is satisfied (maximum number of iterations, solution good enough, etc.):
   - Try to improve the initial solution(s) with respect to some optimization criterion.
3. Determine the best solution of the previous step, and return it as output.

Two of the most frequently applied metaheuristic strategies are genetic algorithms [124] and simulated annealing [97], as they can easily be adapted to many different problems. There also exist popular newer approaches like the ant colony algorithm [49] or particle swarm optimization [95]. Besides, Glover's tabu search [69] has established as another global optimization strategy. We will briefly introduce each of the mentioned strategies in the subsequent sections. Naturally, there exist many more approaches, such that we have to confine ourselves to the most common ones. Recent techniques are introduced in [65] while [104] gives a general overview on metaheuristics.

For discrete optimization problems with linear objective function F there also exist efficient algorithms like the simplex method [43] and interior point approaches [92] that can be used for computation of an approximatively optimal solution. However,

as already mentioned, these algorithms put demands on the shape of the objective function. As we consider more general problem types in this work, we will not go into this special case, but instead refer to textbooks on linear optimization like [64].

2.2.2 Simulated annealing

The *simulated annealing algorithm* is a global optimization heuristic that is derived from the physical process of annealing in solids, where the objective is it to determine the ground state of a system [97]. It is "motivated by the desire to avoid getting trapped in poor local optima, and hence, occasionally allows uphill moves to solutions of higher cost, doing this under the guidance of a control parameter called the temperature" ([87], p.380). Starting from an initial solution, in every iteration a new solution in the neighborhood of the current solution is generated. Afterwards, the values of the objective function for the two solutions are compared. If the new solution is better, then it will be used for the next iteration. If it is worse than the current solution, then it will only be accepted with a certain probability that depends on the current temperature value. With progressing time, the probability of accepting such a move in opposite to the optimization direction gets smaller and finally converges to zero. Consequently, the simulated annealing algorithm is composed of the following steps [80]:

Input:

- Initial temperature T_0.
- Temperature cooling schedule $(T_k)_{k\in\mathbb{N}}$.
- Repetition schedule $(\xi_k)_{k\in\mathbb{N}}$.

Algorithm:

Compute an initial solution S.
$k := 0$.
repeat
  for all $i = 1, 2, \ldots, \xi_k$ **do**
    Generate a solution $S' \in N(S)$.
    if $F(S') - F(S) \leq 0$ **then**
      $S := S'$.
    else
      Draw $u \in [0, 1]$ uniformly distributed.
      if $u < \exp\left((F(S) - F(S'))/T_k\right)$ **then**
        $S := S'$.
      end if
    end if

end for
$k = k + 1$.
until some termination criterion is met.

Output:

- The best found solution S.

Remarks:

- Obviously, the simplest case of a cooling schedule is one where the temperature is decreased by a certain factor $\delta \in]0, 1[$ and then always hold constant for a fixed number of iterations ξ before it is decreased again by δ and so on. By using the notation from above, this corresponds to $T_k := T_0 \cdot \delta^k$ and $\xi_k = \xi$ for all $k = 1, 2, \ldots$. We will always refer to this simple case if anywhere in this work there is talk of "classic simulated annealing" or its abbreviation CSA.
- The initial solution, which is required as input for the simulated annealing algorithm, is randomly generated for most applications.
- The neighbored solution $S' \in N(S)$ is drawn randomly with equal distribution over all possible neighbors.
- As a consequence of the previous remark, the neighborhood of a state S has to be specified beforehand. For optimization problems with continuous solution space this can simply be done by setting $N(S) := \{S' \mid d(S, S') < \varepsilon\}$ for some fixed value $\varepsilon > 0$ and appropriate distance measure d. For discrete optimization problems, however, a distance measure is usually not available and thus, the neighborhood must be defined problem specific.

2.2.3 Genetic algorithms

Genetic algorithms are a class of optimization strategies that are inspired by evolutionary processes in biology. In the literature, John Holland [82] and some of his students are mentioned as originator of genetic algorithms [139]. They adapt the reproduction of organisms, where changes take place by the principles of *selection*, *mutation*, and *crossover*. The connection to optimization problems is established by the term of *fitness* of a population. A genetic algorithm "allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the fitness" ([78], p.22). Consequently, every individual of the population is a solution, and its fitness corresponds to the value of the objective function.

Regarding the selection process, the solutions of the current population generation are compared with respect to their fitness (value of the objective function). Afterwards, the "best" individuals are selected and recombined within the crossover

process. As to that, in each case two individuals are combined, and new individuals are created that take characteristics of both parental units. So, in other terms, two given solutions of an optimization problem have to be combined in an appropriate way to create a new solution. Then, each new solution is randomly changed within the mutation process, and the corresponding fitness value is recorded. The resulting new generation is finally used as input for the next iteration of selection, crossover, and mutation. This whole process is repeated until the population has converged, which means that the fitness values of the individuals contained in this population do not differ significantly.

So in conclusion, a genetic algorithm consists of the following abstract steps that have to be specified for a concrete practical application (taken from [17]):

Input:

- Population size p_{size}.

Algorithm:

Generate initial population.
Compute fitness of each individual.
repeat
  for all $i = 1, 2, \ldots, p_{size}/2$ **do**
    Select two individuals from old generation for mating.
    Recombine the two individuals to give two offspring.
    Mutate characteristics of the offspring.
    Compute fitness of the two offspring.
    Insert offspring in new generation.
  end for
until population has converged.

Output:

- The fittest individual.

2.2.4 Particle swarm optimization

The *particle swarm optimization* (PSO) algorithm was invented by Kennedy and Eberhardt as "a concept for the optimization of nonlinear functions using particle swarm methodology" ([95], p.1942). It is inspired from social behaviour simulation, and there is a certain relationship to genetic algorithms [54].

The central idea of PSO is to create an initial population of particles (which correspond to initial solutions of the optimization problem) that randomly fly around the search space (which is assumed to have dimension D) towards better solutions. In doing so, the currently best known *individual* position of every every particle i (denoted by $\vec{p}_i$) is stored as well as the corresponding fitness (value of the objective function $pbest_i$). Moreover, the best known position $\vec{p}_g$ and fitness over *all* particles *gbest* is also recorded. Then, "the particle swarm optimization concept consists of, at each time step, changing the velocity (accelerating) each particle i toward its $pbest_i$ and *gbest* locations. Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward $pbest_i$ and *gbest* locations." ([55], p.81).

The original PSO implementation finally looks as follows [114] ($\vec{x}_i$ is the current position and $\vec{v}_i$ the velocity of the i^{th} particle):

Input:

- Number of particles n.
- Velocity distribution bounds $\varphi_1, \varphi_2 > 0$.
- Lower and upper bound for the velocity $V_{min} < 0$ and $V_{max} > 0$.

Algorithm:

Initialize a population array of n particles with random positions and velocities on D dimensions in search space.
repeat
For each particle, evaluate the desired optimization fitness function in D variables.
Compare particle's fitness evaluation with its $pbest_i$. If current value is better than $pbest_i$, then set $pbest_i$ equal to the current value, and $\vec{p}_i$ equal to the current location $\vec{x}_i$ in D-dimensional space.
Identify the particle in the neighborhood with the best success so far, and assign its index to the variable g.
Change the velocity and position of the particle according to the following equations:

$$\vec{v}_i := \vec{v}_i + \vec{U}(0, \varphi_1) \cdot (\vec{p}_i - \vec{x}_i) + \vec{U}(0, \varphi_2) \cdot (\vec{p}_g - \vec{x}_i),$$
$$\vec{x}_i := \vec{x}_i + \vec{v}_i.$$

until some termination criterion is met.

Output:

- The fittest particle.

Remark:

- The velocity of each particle i has to be kept within the fixed range, $\vec{v}_i \in [V_{min}, V_{max}]$.

2.2.5 Ant colony algorithm

Another biologically inspired optimization heuristic is the *ant colony optimization algorithm* (ACO) from Dorigo [49]. This algorithm imitates the process of food provision of ant colonies. The final objective within this process is to find the shortest path between the location of the ant colony and a source of food. ACO implementations are thus especially suited for graph based problems like the travelling salesman problem [52] or vehicle routing problems [48].

In nature, ants emit a specific secretion called *pheromone* to mark their walking trails. After some time, however, the pheromone evaporates. The crucial point for successful food acquirement is now the fact that other ants follow trails that are intensely marked with a higher probability than trails that are marked less or even unmarked. "As a result, the collective behaviour that emerges is a form of positive feedback loop where the probability with which an ant chooses a path increases with the number of ants that previously chose the same path" ([144], p.351). So finally, the ants will determine the shortest path between the colony and the food source as a consequence of this collective behaviour [51].

To apply the above concept on combinatorial optimzation problems, specific operations for initialization of the pheromone values as well as for their update have to be implemented. Furthermore, a suited procedure for construction of new solutions with respect to the current pheromone values has to be designed. Optionally, an additional local search step can be integrated to improve a recently constructed solution. The ACO procedure then has the following structure [50]:

Input:

- Ant colony population size n.

Algorithm:

Initialize vector of pheromone values $\mathcal{T}$.
$S_{best} := NULL$.
while termination conditions not met **do**
  $\mathcal{S}_{iter} := \emptyset$.
  for all $j = 1, 2, \ldots, n$ **do**

Construct a new solution S_{new} with respect to $\mathcal{T}$.
if S_{new} is a valid solution **then**
Try to improve S_{new} by performing local search (optional).
if $(F(S_{new}) < F(S_{best}))$ or $(S_{best} = NULL)$ **then**
$S_{best} := S_{new}$.
end if
$\mathcal{S}_{iter} := \mathcal{S}_{iter} \cup \{S_{new}\}$.
end if
end for
Apply pheromone update with respect to $\mathcal{T}$, $\mathcal{S}_{iter}$ and S_{best}.
end while

Output:

- The best found solution S_{best}.

2.2.6 Tabu search

The last metaheuristic we will discuss in this chapter is the *tabu search algorithm* from Glover [67], [68]. Similar to simulated annealing, this method "can be viewed as an iterative technique which explores a set of problem solutions [...] by repeatedly making moves from one solution S to another solution S' located in the neighborhood $N(S)$ of S" ([70], p.4). In contrast to the other metaheuristics introduced so far, tabu search uses a list where the last visited solutions up to a fixed length l_{max} are stored. Within every iteration, the next computed solution must not be contained in the list. If the length of this tabu list reaches the maximum feasible length l_{max}, then the "oldest" solution is erased from and the new solution is added to the list in every subsequent iteration. The strategy for selecting the next neighbored solution is not predefined and can be arbitrarily chosen.

The particular steps of the basic tabu search algorithm can be identified as follows [104]:

Input:

- Maximum length of the tabu list l_{max}.
- Number of checked neighbors n of each intermediate solution.

Algorithm:

Compute initial solution S_{init}.
$S_{best} := S_{curr} := S_{init}$.

Initialize empty queue $\mathcal{Q}$ (corresponds to the tabu list).
Enqueue S_{curr} into $\mathcal{Q}$.
repeat
  if $Length(\mathcal{Q}) > l_{max}$ **then**
    Remove oldest element from $\mathcal{Q}$.
  end if
  Compute new solution $S_{new} \in N(S_{curr})$.
  $S_{opt} := S_{new}$.
  for all $i = 1, 2, \ldots, (n-1)$ **do**
    Compute new solution $S_{new} \in N(S_{curr})$.
    if $S_{new} \notin \mathcal{Q}$ and $(F(S_{new}) < F(S_{opt})$ or $S_{opt} \in \mathcal{Q})$ **then**
      $S_{opt} := S_{new}$.
    end if
  end for
  if $S_{opt} \notin \mathcal{Q}$ and $F(S_{opt}) < F(S_{curr})$ **then**
    $S_{curr} := S_{opt}$.
    Enqueue S_{opt} into $\mathcal{Q}$.
  end if
  if $F(S_{curr}) < F(S_{best})$ **then**
    $S_{best} := S_{curr}$.
  end if
until some termination criterion is met.

Output:

- The best found solution S_{best}.

2.2.7 Drawbacks of existing approaches

As already mentioned in chapter 1, many approaches have the problem that either the number of algorithm specific operators which have to be defined or the number of input parameters which have to be fixed in advance is quite high. Genetic algorithms, for example, require the definition of three additional operators, namely mutation, crossover, and selection. The other two terms, individuals and fitness, which correspond to the solution space and the objective function as well as a termination criterion have to be defined for all global optimization heuristics.
On the other side, classic simulated annealing and particle swarm optimization each require only the definition of one additional operator, the neighborhood and the velocity, respectively. However, for an application of particle swarm optimization, in turn, five input parameters have to be set by the user (total number of particles, lower and upper boundary of the velocity, and two parameters for the velocity

change distribution), whereas classic simulated annealing only has three of them (initial temperature, temperature degression coefficient, and number of runs with constant temperature).
One other characteristic that is shared by all approaches introduced above except for simulated annealing and tabu search is that the number of newly generated and processed solutions is very high in general (depending on the number of particles in case of particle swarm optimization and the population size for genetic and ant colony algorithms). For the application of tabu search, on the other hand, many solutions must be compared in every iteration (depending on the length of the tabu list). In any case, these conditions require that solutions can be generated and completely evaluated efficiently. For complex combinatorial optimization problems with many side conditions this is not invariably the case as we will see later in chapter 4 when we discuss the CASASSIGN problem class, which is a special case of a coupled earliness tardiness job shop scheduling problem.
Simulated annealing, on the contrary, only works with one specific solution at any time which is just slightly modified in each iteration. Thus, except for the initial state, which is required as input, no additional solutions have to be generated. Furthermore, in many cases and also when applied to the CASASSIGN problem class, the change in the objective function value after one iteration can be derived without having to re-evaluate the whole solution.
The crucial drawback of classic simulated annealing is the high dependence of the solution quality on the choice of the input parameters as already mentioned in chapter 1 (a survey on research papers regarding this issue is given in chapter 9). Furthermore, classic simulated annealing does not presuppose any knowledge about shape and analytical structure of the objective function: Every neighborhood state has equal probability of being chosen as next one for getting "checked", before a final decision about acceptance of the new state is made (see remarks about the simulated annealing algorithm in section 2.2.2). However, in many applications it is possible to make a point about the probability of each neighbored state being "better" than the current state with respect to the objective function value without having to evaluate it. This is especially useful when evaluating the objective function is very resource-intensive.
To overcome the problems described in the previous paragraph, we will therefore introduce some modifications to the classic simulated annealing algorithm at two specific positions.
First of all, experience shows that for most combinatorial optimization problems it is possible to design an efficient greedy strategy that computes an approximatively optimal solution of this problem. Hence, as first modification of CSA, it is obvious to use the greedy result instead of a randomly generated solution of an optimization problem as initial solution for the simulated annealing algorithm. Thus, the simulated annealing procedure will start from a point that is usually more close to the

globally optimal result than for the classic version, where random initialization is used.
Secondly, we suggest to modify the candidate generation probabilities from an equal distribution to one that takes the analytical structure of the underlying problem into account. We will explain how this task can be accomplished in detail in the next chapter. Moreover, we will give a general construction guidance how these modified candidate generation probabilities can be derived from a greedy evaluation map that evaluates extensions of partial solutions as described in section 2.1.

3 Greedy with modified simulated annealing

Based on the preliminary considerations from last section, we will now introduce our GWMSA strategy in detail. We start with an abstract specification of GWMSA in section 3.1 and describe an application example to a common combinatorial optimization problem afterwards in section 3.2.

3.1 Abstract specification

In this section we will introduce our new metaheuristical GWMSA approach in detail. As already mentioned above, we will first provide an abstract description of the greedy strategy in the next subsection. Our modifications to the classic simulated annealing algorithm from subsection 2.2.2 will be formally described subsequently.

3.1.1 Greedy

In our context, the greedy algorithm for computation of an initial solution for simulated annealing is straightforward. It starts with an empty partial solution S_0 and iteratively adds components $c \in C \setminus S_0$ to this partial solution, such that the value of the evaluation map f is maximized in each step (revise section 2.1 for details about f and its properties). The procedure stops as soon as S_0 forms a complete solution of the given optimization problem, which means $S_0 \in \mathcal{S}$. In detail, this generalized greedy approach consists of the following steps:

Input:

- Set of components C.
- Solution space $\mathcal{S}$.
- Evaluation map f.

Algorithm:

$S_0 := \emptyset$.
while $S_0 \notin \mathcal{S}$ **do**
  $f_{max} := 0$.
  for all $c \in C \setminus S_0 \ : \ S_0 \cup c \in \mathcal{S}_{part}$ **do**
    if $f(S_0, c) > f_{max}$ **then**
      $f_{max} := f(S_0, c)$.
      $c_{max} := c$.
    end if

end for
$S_0 := S_0 \cup c_{max}$.
end while

Output:

- Solution S_0.

The computation time of this general greedy approach is obviously bounded by the product of the size of the solution space and the number of components, thus $\mathcal{O}(|\mathcal{S}| \cdot |C|)$.

The concrete implementations of greedy strategies for our four problem classes introduced later basically comply with the structure of the abstract greedy algorithm specification from above. However, as the individual approaches differ in the way how possible extensions of partial solutions are evaluated, we will provide separate specifications of the greedy strategies for CASASSIGN, TESTSUITE, and TASKALLOC. Only the greedy algorithm used for computation of an initial solution of the WEBRANK problem can be exactly reduced to the structure from above (see section 8.2.1 for details).

3.1.2 Modified simulated annealing

For implementation of simulated annealing, the neighborhood of a solution has to be defined. We will cover this issue first of all in the next subsection, before we describe our central modification of the classic simulated annealing algorithm. Finally, subsection 3.1.2.3 contains the detailed specification of our algorithm.

3.1.2.1 Neighborhood definition

As already described in the remarks of section 2.2.2, for application of the classic simulated annealing algorithm on continuous optimization problems in a metric space, the neighborhood of a point x is generally defined by all points within an ε-environment around x. For combinatorial optimization problems, however, the construction of such an environment is often a non-trivial task due to their discrete characteristics. The definition of a neighborhood is therefore very problem specific, and a general guidance for its construction cannot be given. Usually, the neighborhood of a solution will be chosen as small as possible, such that still all feasible solutions can be reached directly or indirectly via the neighborhood relation.
For many assignment problems like CASASSIGN and TASKALLOC, when replacing one component of a solution the resulting set is a new solution. Even more, the

following condition holds:
For every pair of solutions $S, S' \in \mathcal{S}$ there exist solutions $S_0, S_1, \ldots, S_M \in \mathcal{S}$ and components $c_{00}, c_{01}, c_{10}, c_{11}, \ldots, c_{(M-1)0}, c_{(M-1)1} \in C$, such that

$$S = S_0, S' = S_M \quad \text{and} \quad S_i = (S_{i-1} \setminus c_{(i-1)0}) \cup c_{(i-1)1}$$

for $i = 0, 1, \ldots, (M-1)$.
Then, we can define the neighborhood of a solution S by the set of all other solutions that can be obtained from S by replacement of exactly one component:

$$N(S) := \{S' \in \mathcal{S} \mid \exists I, J \in \{1, 2, \ldots, |C|\} \,:\, S' = (S \setminus c_I) \cup c_J\} \tag{3.1}$$

for every $S \in \mathcal{S}$.

3.1.2.2 Modified candidate generation probabilities

In the classic version of the simulated annealing algorithm [97], every neighbor $S' \in N(S)$ of a solution $S \in \mathcal{S}$ is drawn with the same probability within one iteration. In contrast to that, our modified variant of simulated annealing uses candidate generation probabilities that are adjusted in a way that component replacements which lead to better neighbored solutions with respect to the optimality criterion are preferred.
For combinatorial optimization problems where a greedy strategy with evaluation map f is available that has the properties (i.) and (ii.) described in section 2.1, the construction of these candidate generation probabilities can be accomplished as follows:

Given a solution $S \in \mathcal{S}$, the probability for choosing a neighbored solution $S' \in N(S)$, $\mathbb{P}_{cg}(S, S')$, is composed of the probabilities for excluding a subset of solution components $U = (S \setminus S') \cup V$ with $V \subset S \cap S'$ and including the corresponding complementary set $\bar{U} = S' \setminus (S \setminus U)$. We will derive reasonable exclusion and inclusion probability distributions,

$$\mathbb{P}_{ex} \,:\, \mathcal{S} \times \mathcal{P}(C) \to [0, 1] \text{ and } \mathbb{P}_{inc} \,:\, \mathcal{S}_{part} \times \mathcal{P}(C) \to [0, 1],$$

that favor replacements which will more probably lead to good solutions.

It is clear that removing component subsets $U \subset S$ of a solution $S \in \mathcal{S}$, whose contribution with respect to the value of the evaluation map f is small, will more probably lead to better solutions with respect to the optimality criterion than removing components with higher contribution value. Therefore, we define the exclusion probabilities by

$$\mathbb{P}_{ex}(S,U) := \begin{cases} \frac{g(U)}{\sum_{\tilde{U}\in\mathcal{C}_S} g(\tilde{U})}, & \text{if} \quad U \in \mathcal{C}_S \\ 0, & \text{else} \end{cases}, \tag{3.2}$$

where

$$\mathcal{C}_S := \{U \subset S \,|\, \exists\, S' \in N(S) \,:\, U = S \setminus (S \cap S')\}$$

and $g(U) := \frac{1}{f(S\setminus U,U)}$. As we use the inverse values of the evaluation map, $\mathbb{P}_{ex}(S,U)$ will be be greater for those $U \in \mathcal{C}_S$ whose evaluation value contribution (given by $f(S \setminus U, U)$) is small. Note that the denominator of $g(U)$ is greater than zero for every $U \in \mathcal{C}_S$ as $S \in \mathcal{S}_{part}$ by definition.

Vice versa, let $S_{part} \in \mathcal{S}_{part}$ and $U \subset C \setminus S_{part}$. Then, we set the inclusion probabilities as follows:

$$\mathbb{P}_{inc}(S_{part},U) := \begin{cases} \frac{f(S_{part},U)}{\sum_{V\,:\,(S_{part},V)\in\mathcal{D}} f(S_{part},V)}, & \text{if} \quad (S_{part},U) \in \mathcal{D} \\ 0, & \text{else} \end{cases}, \tag{3.3}$$

where

$$\mathcal{D} := \{(Y,Z) \in \mathcal{S}_{part} \times \mathcal{P}(C) \,|\, \exists S \in \mathcal{S}, S' \in N(S) \,:\, Y = S \cap S' \wedge S' = Y \cup Z\}.$$

By construction, extensions $U \subset C \setminus S_{part}$ of a partial solution $S_{part} \in \mathcal{S}_{part}$ with high evaluation value f are chosen more probably than others.

Hence, for two arbitrary solutions $S, S' \in \mathcal{S}$, if S is the current solution, then the probability for choosing S' as next solution candidate is defined by

$$\mathbb{P}_{cg}(S,S') := \begin{cases} \mathbb{P}_{ex}(S, S \setminus (S \cap S')) \cdot \mathbb{P}_{inc}(S \cap S', S' \setminus (S \cap S')), & \text{if} \quad S' \in N(S) \\ 0, & \text{else} \end{cases}.$$

It is clear that the above definition of candidate generation probabilities usually is impracticable if all possible solutions are defined to be neighbored to each other. The reason is that the solution set $\mathcal{S}$ will be very large in general and thus, the computational effort for determination of the probabilities is very high. Hence, the neighborhood relation $N(S)$ of a solution S has to be narrowed down, such that the null space of $\mathbb{P}_{cg}$ is as great as possible. However, as already mentioned in the previous subsection, one has to make sure that every solution is still reachable (directly or indirectly) by every other solution via the neighborhood relation.
For combinatorial optimization problems with neighborhood definition (3.1) the above definitions of inclusion and exclusion probabilities reduce to

$$\mathbb{P}_{ex}(S,c) := \frac{g(c)}{\sum_{\tilde{c}\in S} g(\tilde{c})}$$

for every $c \in S$, where $g(c) := \frac{1}{f(S \setminus c, c)}$ and

$$\mathbb{P}_{inc}(S_{part}, c) := \frac{f(S_{part}, c)}{\sum_{\tilde{c} \in C \setminus S_{part}} f(S_{part}, \tilde{c})}$$

for every $c \in C \setminus S_{part}$. Thus, for problems of this type we obtain

$$\mathbb{P}_{cg}(S, S') := \begin{cases} \mathbb{P}_{ex}(S, c_S) \cdot \mathbb{P}_{inc}(S \setminus c_S, c_{S'}), & \text{if} \quad \exists\, c_S, c_{S'} \in C \,:\, S' = (S \setminus c_S) \cup c_{S'} \\ 0, & \text{else} \end{cases}$$

for every $S, S' \in \mathcal{S}$.

3.1.2.3 Algorithm specification

In each modified simulated annealing iteration a subset of components $C_1 \subset S_{curr}$ of the current solution S_{curr} is drawn and randomly replaced by another subset of components C_2, such that a new complete solution $S_{new} \in N(S_{curr})$ is formed. The candidate generation of S_{new} from S_{curr} is done with respect to the probability distribution $\mathbb{P}_{cg}$ from above. Afterwards, the new solution S_{new} is evaluated and accepted or withdrawn in the same way as for classic simulated annealing, see subsection 2.2.2. These two steps are repeated until no more significant improvements in the objective function F are achievable. The preliminary considerations from above yield to the following optimization procedure:

Input:

- Solution space $\mathcal{S}$.
- Initial solution $S_{init} \in \mathcal{S}$.
- Set of components C.
- Objective function F.
- Candidate generation probability distribution $\mathbb{P}_{cg}$.
- Initial temperature T_{init}.
- Number of runs with constant temperature ξ.
- Temperature degression coefficient δ.
- Termination threshold ε.

Algorithm:

$F_{curr} := F(S_{init})$, $S_{curr} := S_{init}$, $F_{best} := F_{curr}$,
$S_{best} := S_{curr}$, $T := T_{init}$.
repeat
  $F_{old} := F_{best}$.
  for all $i = 1, 2, \ldots, \xi$ **do**

Draw $S_{new} \in N(S_{curr})$ with probability $\mathbb{P}_{cg}(S_{curr}, S_{new})$.
Compute $F_{new} := F(S_{new})$.
Draw $u \in [0, 1]$ uniformly distributed.
if $\exp((F_{curr} - F_{new})/T) > u$ **then**
$S_{curr} := S_{new}$.
$F_{curr} := F_{new}$.
end if
if $F_{new} < F_{best}$ **then**
$S_{best} := S_{new}$.
$F_{best} := F_{new}$.
end if
end for
$T := T \cdot \delta$.
until $|F_{old} - F_{best}| < \varepsilon$.

Output:

- S_{best}.

Note that in comparison to the classic simulated annealing algorithm from section 2.2.2 we have added another solution variable S_{best} (and a corresponding variable F_{best} for the value of the objective function) that always carries the best solution found so far. This slightly modified variant has shown better results in practical applications than the classic version [104].

In the following chapters we design implementations of the modified simulated annealing algorithm for the four problem classes CASASSIGN, TESTSUITE, TASKALLOC, and WEBRANK. This includes the definition of a suited neighborhood relation and modified candidate generation probabilities for each class. Afterwards, the abstract algorithm specification from above is applicable for every problem class by simply replacing the corresponding definitions for neighborhood relation and modified candidate generation probabilities.

3.2 Application example: Set covering problem

Two of the four problem classes that will be introduced later are generalizations of the *set covering problem* [93]. We will therefore demonstrate the applicability of our concepts from last section for this problem instance first. We will start with a formulation of set cover as a binary integer program in section 3.2.1 and introduce a greedy strategy in section 3.2.2. The neighborhood as well as modified candidate generation probabilities for the modified simulated annealing algorithm are defined in sections 3.2.3 and 3.2.4.

3.2.1 Set cover as binary integer program

For practical implementations, it is common to formulate the minimum set covering problem as a binary integer program as follows [36]:
Let $\mathcal{U}$ be a universe of different elements and C be a family of subsets of $\mathcal{U}$. In the following, we always will take for granted that $\mathcal{U} = \bigcup_{c \in C} c$ (otherwise the set covering problem would have no feasible solution). Next, we associate a binary variable $x_c \in \{0, 1\}$ to each $c \in C$. Then, finding an optimal solution of the minimum set covering problem is equivalent to solving the following binary integer program:

- Minimize

$$F(C) := \sum_{c \in C} \lambda(c) \cdot x_c, \tag{3.4}$$

- such that

$$\sum_{c \,:\, e \in c} x_c \geq 1 \quad \forall\, e \in \mathcal{U}. \tag{3.5}$$

So, the task is to find a subset $S \subset C$, such that the sum of $\lambda(c)$ for all $c \in S$ (given by (3.4)) is minimized, and every element $e \in \mathcal{U}$ is contained in at least one set $c \in S$ (given by inequalities (3.5)), coincidently.

As we can identify S with the set $\{c \in C \,:\, x_c = 1\}$ in anology to (3.4), and for ease of notation we will use

$$F(S) := \sum_{c \in S} \lambda(c). \tag{3.6}$$

The side conditions in (3.5) can be translated to

$$|\{c \in S \,:\, e \in c\}| \geq 1 \quad \forall\, e \in \mathcal{U}. \tag{3.7}$$

Obviously, with respect to our abstract problem definition from subsection 2.1, the set of components C can be identified with the given family of subsets of $\mathcal{U}$, and every component $c \in C$ corresponds to a subset. Consequently, the set of solutions $\mathcal{S}$ contains all those collections of subsets $S \subset C$ where every element $e \in \mathcal{U}$ is covered.

3.2.2 Greedy strategy for set cover

Chvatal has introduced a greedy strategy for the set covering problem in [36] which was later proven to be the best-possible polynomial time approximation algorithm

for set cover [2].
Let

$$\kappa(e, S) := |\{c \in S \,:\, e \in c\}| \tag{3.8}$$

be the *coverage* of an element $e \in \mathcal{U}$. The coverage of an element e is the number of sets c in S where e is contained. Then, for every partial solution $S_0 \subset C$ we can define

$$d(S_0, c) := |\{e \in \mathcal{U} \,:\, e \in c \wedge \kappa(e, S_0 \setminus c) = 0 \wedge \kappa(e, S_0 \cup c) = 1\}|.$$

Descriptively, for a set $c \in S$, $d(c)$ is the number of elements $e \in \mathcal{U}$ that are exclusively covered by c. Finally, we set

$$f(S_0, c) := \frac{d(S_0, c)}{\lambda(c)} \tag{3.9}$$

for every $c \in S$ and every partial solution S_0. Note that f satisfies the conditions (i.) and (ii.) from section 2.1 and can thus be interpreted as greedy evaluation map. By using the definitions of C, $\mathcal{S}$ and f from above, Chvatal's algorithm has now exactly the form of the abstract greedy strategy from subsection 3.1.1.

3.2.3 Neighborhood definition

Given a feasible, not necessarily optimal set cover S that covers all elements $e \in \mathcal{U}$, a neighbor solution S' of S is generated by removing exactly one set $c \in S$ from S and replacing it by as many other sets $\tilde{c} \in C \setminus S$ as necessary to obtain a set cover S' that completely covers $\mathcal{U}$ again:

$$N(S) := \Bigg\{ S' \,\Bigg|\, \bigcup_{c' \in S'} \bigcup_{e \in c'} e = \mathcal{U}, \exists c \in S \setminus S', c_1, \ldots, c_n \in S' \setminus S \,:\, S' = (S \setminus c) \cup c_1 \cup \ldots \cup c_n \Bigg\}.$$

3.2.4 Definition of candidate generation probabilities

Regarding our neighborhood definition from last subsection, we have to define probabilities for the inclusion and exclusion of sets into and from a set cover.

The denominator of the evaluation map f in (3.9) can never become zero, as we assume that every set contains at least one element (otherwise this set would be useless and could be excluded from C without loss) and thus, $\lambda(c) \geq 1$. So, refering to subsection 3.1.2.2, for all $c \in C \setminus S_0$ we can set

$$\mathbb{P}_{inc}(S_0, c) := \frac{f(S_0, c)}{\sum_{\tilde{c} \in C \setminus S_0} f(S_0, \tilde{c})}. \tag{3.10}$$

For the definition of exclusion probabilties, we have to make a case differentiation: When considering

$$g(S_0, c) := \frac{1}{f(S_0, c)} = \frac{\lambda(c)}{d(S_0, c)}, \tag{3.11}$$

it may happen that the denominator of (3.11) becomes zero if $d(S_0, c) = 0$. This phenomenon can be explained as follows:
Let c_1 be a set that has been included to a partial solution S_0 in a previous step. Now suppose that in two subsequent steps two sets c_2 and c_3 with $d(S_0, c_2) > 0$, $d(S_0, c_3) > 0$, and $c_2 \cup c_3 \supsetneq c_1$ have been included to S_0. Then, necessarily $d(S_0, c_1) = 0$, c_1 can be removed from S_0, and all elements $e \in \mathcal{U}$ will still be covered. Let

$$\Gamma_0 := \{c \in S_0 \, : \, d(S_0, c) = 0\}.$$

If the above case occurs, it is obvious to draw and exclude as many $c \in \Gamma_0$ as necessary until $\Gamma_0 = \emptyset$ again. Furthermore, it is clear that the exclusion probability of sets $c \in \Gamma_0$ should be proportional to $\lambda(c)$, as the removal of longer, redundant sets from S_0 will reduce the objective function F from formula 3.6 more significantly. Thus, for the special case that $\Gamma_0 \neq \emptyset$ we define exclusion probabilities as follows:

$$\begin{aligned} &\mathbb{P}_{ex}(S_0, c) := \frac{\lambda(c)}{\sum_{\tilde{c} \in \Gamma_0} \lambda(\tilde{c})}, \quad \text{if} \quad c \in \Gamma_0 \quad \text{and} \\ &\mathbb{P}_{ex}(S_0, c) := 0, \quad \text{else.} \end{aligned} \tag{3.12}$$

For the regular case that $\Gamma_0 = \emptyset$ we can define the exclusion probability distribution analogously to the inclusion probabilities by

$$\mathbb{P}_{ex}(S_0, c) := \frac{g(S_0, c)}{\sum_{\tilde{c} \in S_0} g(S_0, \tilde{c})}. \tag{3.13}$$

Altogether, we obtain the following definition of candidate generation probabilities:

$$\mathbb{P}_{cg}(S, S') := \begin{cases} \frac{\lambda(c)}{\sum_{\tilde{c} \in \Gamma_0} \lambda(\tilde{c})}, & \text{if} \quad c \in \Gamma_0 \wedge S' = S \setminus c \\ 0, & \text{else} \end{cases},$$

if $\Gamma_0 \neq \emptyset$ and

$$\mathbb{P}_{cg}(S, S') := \begin{cases} \mathbb{P}_{ex}(S, c) \cdot \prod_{i=1}^{n} \mathbb{P}_{inc}((S \setminus c) \bigcup_{j=1}^{i-1} c_j, c_i) & \text{if} \quad S' = (S \setminus c) \bigcup_{i=1}^{n} c_i \\ 0, & \text{else} \end{cases},$$

if $\Gamma_0 = \emptyset$.

If Γ_0 is empty, the above definition ensures that one set $c_0 \in S_0$ is drawn and randomly replaced by as many other sets $c_i \in C \setminus S_0$ as necessary to obtain a new feasible solution S_1 (that is $\kappa(e, S_1) \geq 1$ for all $e \in \mathcal{U}$). Otherwise, sets $c \in \Gamma_0$ are drawn and excluded until $\Gamma_0 = \emptyset$ again.
Note that for the regular case ($\Gamma_0 = \emptyset$), the definitions of the inclusion and exclusion probabilities from formulas (3.10) and (3.13) exactly follow the abstract structure from (3.3) and (3.2) in subsection 3.1.2.2.

Hence, for implementation of the modified simulated algorithm for the set covering problem, a case differentiation whether Γ_0 is empty or not, has to be integrated additionally into the abstract specification from subsection 3.1.2.3, before a neighbor solution candidate is drawn.

4 Casualty assignment problem

4.1 Problem specification

As already mentioned in chapter 1, we will distinguish between a static problem definition of CASASSIGN that considers the planning of security measures for a mass event that will take place in the future and an online version, where we assume that we are in a situation when a major incident during a mass event has just occurred.

4.1.1 Static assignment

4.1.1.1 Introduction

In recent years, the number and frequency of major events such as concerts, demonstrations, and sport events has considerably increased. Additionally, the average number of attendees is still growing (e.g., the audience of football games [47] or rock concerts). Apparently, the high number of attendants carries a tremendous risk. Apart from the danger of terrorist attacks, factors that at first glance appear trivial can cause a mass panic with many injured people. Today, units involved in the treatment of injured such as police, fire brigade, or physicians are frequently pushed to their limits (e.g., Love Parade disaster in 2010 with 21 victims [130] or the mass panic in the Heysel Arena [16] in 1985 with 39 victims). Clearly, the preparation and planning of safety measures for mass events has gained in importance. In order to medicate all casualties as good and quickly as possible, the transport and waiting times have to be minimized. In addition, the casualties need medical care by qualified personnel. One key factor of safety measures is the allocation of transport vehicles as well as capacities in hospitals. A determination of adequate demands before a specific mass event is non-trivial, and often the allocated capacities turn out to be not sufficient. The reasons for these deficits are twofold. First, organizers aim at a high profit. Thereby, they risk a poor treatment in case of unlikely accidents. Legal restrictions and simulations can help to address this problem. Second, the assignment of casualties to transport vehicles and hospitals is not optimized these days leading to an overall unbalanced workload. Nowadays, the selection of a hospital per injured is usually taken ad-hoc, sometimes even after the transport has already been started. For that purpose, the injured are categorized into injury groups. Patients with severe injury will be preferably treated. However, discussions with the chief officers of the health department at a city council in Germany have motivated the need for a more detailed classification. Often, a longer waiting time will affect the patients' state of health leading to a falsified prioritization. To

improve the process, we propose the definition and consideration of functions over time for all injuries reflecting the corresponding course of disease. These functions assign a penalty value to every casualty depending on its injury and waiting time until medication.

Figure 4.1 Care of injured and crowd at music event. The high number of injured and difficult access to them poses new challenges for involved units.

We will incorporate the penalty functions into a general mathematical model to optimize the assignment of casualties to transport vehicles and physicians in hospitals. Moreover, we impose the constraint that each casualty can be assigned only to those physicians who are able to medicate the specific type of injury the casualty suffers from. Before describing our mathematical model in detail, we first want to motivate the use of penalty functions in the next subsection.

4.1.1.2 Triage groups and penalty functions

If nowadays a high number of casualties have to be medicated at once, the injured are first divided into different classes (*triage categories*) to ensure an adequate supply. Patients with a higher priority are served first. These groups differ only slightly from country to country. [140] give a detailed overview about those differences. For the purpose of this work, we use the definition of [41]:

- *T1: Urgent vital thread*, e.g., respiratory insufficiency with asphyxia, massive (external) bleeding, heavy shock, serious thorax, or abdomen trauma, face and/or respiratory system burnings, tension pneumothorax.
  → Immediate treatment necessary.

- *T2: Seriously injured*, e.g., craniocerebral injury, serious eye and/or face injury, instable luxation, exposed fracture/joint.
  → Postponed treatment urgency, supervision.

- *T3: Slightly injured*, e.g., uncomplicated fracture, repositionable luxation, contusion, distortion, graze.
  → Posterior treatment (ambulant if applicable).

- *T4: Without any viability chance, dying*, e.g., open craniocerebral injury with cerebral mass discharge, severest, and extended burnings.
  → Supervising, death awaiting treatment, terminal care.

Often, a longer waiting time will affect the patients' state of health, because the severity of many injuries depends on the therapy time. A tension pneumothorax, for example, is nonhazardous as far as treated immediately. But already after a short period of time without treatment, a tension pneumothorax can get life-threatening. Note that applying the concept of triage categories as described above fixes a priority for each casualty not considering the waiting time. Therefore, assignment switches are not allowed and often leading to a falsified prioritization.

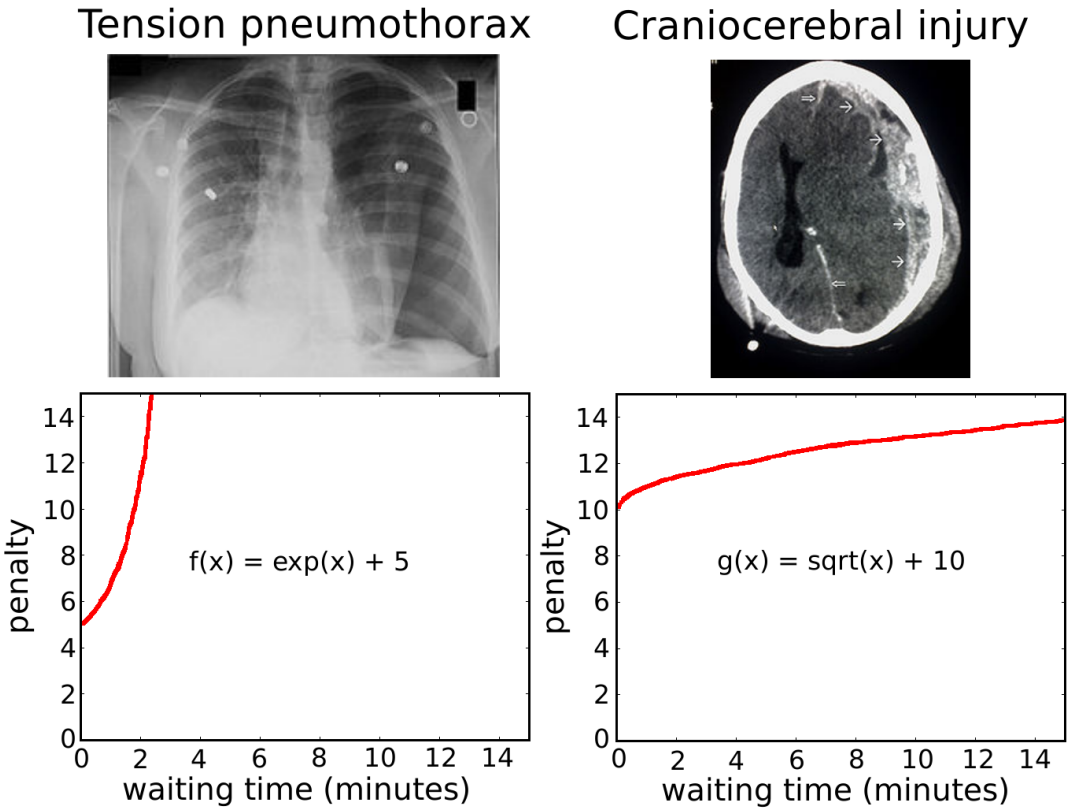


Figure 4.2 Example pictures and functions illustrating the course of disease in time

To improve the process, our optimization method considers functions (*penalty functions*) over time for all injuries describing the corresponding course of injury. The

penalty functions π_ι assign a penalty value to each injury $\iota \in I$ (set of injuries) depending on the waiting time t until medication. Figure 4.2 illustrates the course of injury for a tension pneumothorax (left figure) and a craniocerebral injury (right figure). Obviously, these two diseases have a different initial priority and course. A long waiting time thus has a different impact. In case of pneumothorax, the damage for quick treatment is very low but increases exponentially in time. Indeed, the waiting time for a craniocerebral injury also influences the damage but has nowhere the effect as for a tension pneumothorax.

4.1.1.3 Optimization problem and criteria

Figure 4.3 illustrates the optimization problem described in subsection 4.1.1.1 including relationships between involved entities. Intuitively, our aim is to find an optimal assignment between casualties, transport vehicles, and physicians, such that all casualties can be medicated as good and quickly as possible with respect to their individual injuries. In the following, we formally introduce the involved factors. We also define the optimization function which calculates the total damage over all patients. Clearly, the goal is to minimize this function.

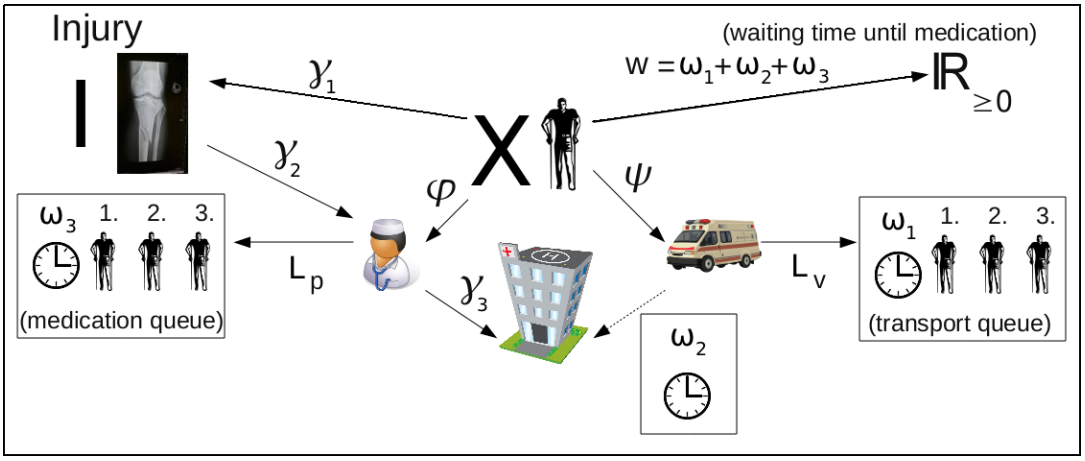


Figure 4.3 Interrelations between involved entities in emergency supply for mass events. Core entities are casualties (X) with injury I mapped to a vehicle and a doctor that, in turn, belongs to a specific hospital. The optimization is performed subject to waiting waiting time w and doctors' qualifications.

Let V, P, and H be a set of vehicles, physicians, and hospitals. Furthermore, let M be a set of assignment tuples (φ, ψ), where φ maps each casualty $\chi \in X$ to a physician $p \in P$ and ψ maps each casualty to a vehicle $v \in V$, respectively. We define following optimization function:

$$F := \sum_{\chi \in X} \pi_{\gamma_1(\chi)}(w(\chi)), \tag{4.1}$$

with

$$w : X \to \mathbb{R}_{\geq 0},\ w(\chi) := \omega_1(\chi) + \omega_2(\chi) + \omega_3(\chi),\ \forall\, \chi \in X.$$

The function first considers the total waiting time $w(\chi)$ until medication for casualty χ. Subsequently, this waiting time serves as a parameter for the penalty function $\pi_{\gamma_1(\chi)}$ describing the course of injury for $\gamma_1(\chi) \in I$ that casualty $\chi \in X$ is suffering from. The overall damage is the sum over all casualties' single damage $\pi_{\gamma_1(\chi)}(w(\chi))$. The waiting time itself, in turn, is the sum of

- $\omega_1 : X \to \mathbb{R}_{\geq 0}$: Waiting time until transport.
- $\omega_2 : X \to \mathbb{R}_{\geq 0}$: Transport duration for each casualty.
- $\omega_3 : X \to \mathbb{R}_{\geq 0}$: Medication waiting time in hospital.

The objective is to find an optimal mapping of casualties to physicians and vehicles. Note that $(\varphi, \psi) \in M$ only assigns casualties to physicians and vehicles but does not define an order of treatment. Therefore, we additionally define the functions $(L_p)_{p \in P}$, $(L_v)_{v \in V}$ with

$$L_p : \varphi^{-1}(p) \to \{1, 2, \ldots, |\varphi^{-1}(p)|\},$$
$$L_v : \psi^{-1}(v) \to \{1, 2, \ldots, |\psi^{-1}(v)|\}.$$

The functions define for all physicians p and vehicles v a sequence of treatment for patients that are either assigned to the specific doctor $(\varphi^{-1}(p))$ or vehicle $(\psi^{-1}(v))$. The mappings (φ, ψ) and orderings $(L_p)_{p \in P}$ and $(L_v)_{v \in V}$ should be chosen, such that they minimize F and concurrently satisfy the following condition:

- C1: A casualty χ is assigned to a qualified physician $\varphi(\chi)$:
  $\varphi(\chi) \in \gamma_2(\gamma_1(\chi)), \forall\, \chi \in X$ with classification function $\gamma_1 : X \to I$ mapping a casualty to an injury, and $\gamma_2 : I \to \mathcal{P}(P)$ assigning injuries to the set of qualified doctors.

Note that the ordering sequences $(L_p)_{p \in P}$ and $(L_v)_{v \in V}$ are bijective, since each casualty has exactly one position in the assigned transport and medication queue. It can be easily seen that in the worst case (each physician $p \in P$ can medicate each type of injury) there are $|P|^n \cdot |V|^n$ different combinations of feasible maps φ and ψ. Then we have at most $n! \cdot n!$ possible bijective orderings for each $(\varphi, \psi) \in M$ if n is the total number of casualties. Thus, altogether we obtain $\mathcal{O}(n^n)$ different combinations. In addition, the optimization problem is non-linear: Obviously, the objective function F from equation (4.1) can contain non-linear penalty functions

$\pi_{\gamma_1(\chi)}$ and thus, using linear optimization schemes like the simplex algorithm [108] or interior point methods [109] for solving our optimization problem is impossible. Therefore, using an approximation heuristic such as simulated annealing to compute an optimal solution of the problem is indicated.

Translated to the abstract problem formulation from section 2.1, every component $c \in C$ of the CASASSIGN problem corresponds to a 5-tuple $(\chi, v, p, L_v(\chi), L_p(\chi))$ consisting of a casualty χ, an ambulance vehicle v, and a physician p where the casualty is assigned to with corresponding queueing positions $L_v(\chi)$ and $L_p(\chi)$. A (feasible) solution $S \in \mathcal{S}$ is a collection of components where every casualty is assigned to an ambulance and a qualified physician in a hospital (formally, $\varphi(\chi) \in \gamma_2(\gamma_1(\chi)$ for every $\chi \in X$). Consequently, the set of solutions $\mathcal{S}$ consists of all feasible assignments. So obviously, the value of the global objective function F from (4.1) is determined by the composition of the current solution S, which means $F = F(S)$.

4.1.2 Online assignment

4.1.2.1 Introduction

In the last section we have considered a static casualty assignment problem. In this context we assumed that the whole set of casualties is known in advance. This assumption is uncritical as we wanted to design a tool for planning security measures in advance to a mass event (e.g. concert, soccer match) and thus *before* a (possible) incident. For simulation purposes, it is therefore appropriate and necessary to estimate the total number of casualties beforehand. However, if this tool should be employed *during* the chaos phase instantly after a major incident has occurred, some adjustments have to be made. One of the challenges is, for example, that in practice, casualties are not registered all at once but usually arrive in small groups. Furthermore in general, the exact number of casualties still to come in the future is unknown in advance.
These additional constraints require a generalization of the formalism and the optimization criterion of the static problem definition from last section in a way that now, total damage over all casualties has to be minimized for every point in time within the temporal horizon of the incident. We will introduce the necessary adaptions in detail in the next subsection.

4.1.2.2 Optimization problem and criteria

The relationships between the involved entities are the same as for the static casualty assignment problem in figure 4.3. Again, our aim is to find an optimal assignment

for casualties X, transport vehicles V, and physicians P, such that all casualties can be medicated as good and quickly as possible with respect to their individual injuries I. Since there exist temporal dependencies related to the arrival time of casualties within our online scenario, this optimization is now executed iteratively not on the whole set of casualties like for the static assignment problem but on small groups. We assume these groups to arrive at intermittent points in time $t_i \in [0, T]$, $i = 1, 2, \ldots, N$ with $T = t_N$ at disaster area. In the following, we formally introduce other involved factors and redefine the optimization function.

Since we now presuppose that the complete set of casualties X is unknown beforehand, we define time-dependent subsets $X(t) \subset X$ for each point in time $t \in [0, T]$, where $X(t)$ defines the set of casualties that have already arrived at disaster area at time t. Obviously, $X(t_i) \subset X(t_j)$ for $0 \leq t_i < t_j \leq T$ and $X = X(T)$, since we defined T as the particular point in time when the last group of casualties arrives at disaster area. Analogously, we define sets $M(t)$ for each point in time $t \in [0, T]$, where $M(t)$ contains all assignment tuples (φ_t, ψ_t) that map all casualties $\chi \in X(t)$ to a physician $p \in P$ and to a vehicle $v \in V$, respectively. Again, we have $M = M(T)$.
We define following band of optimization functions (one function for each $t \in [0, T]$):

$$F_t := \sum_{\chi \in X(t)} \pi_{\gamma_1(\chi)}(w(\chi)), \tag{4.2}$$

with waiting time until medication

$$w : X \to \mathbb{R}_{\geq 0},\ w(\chi) := \omega_0(\chi) + \omega_1(\chi) + \omega_2(\chi) + \omega_3(\chi),\ \forall\, \chi \in X, \tag{4.3}$$

where ω_1, ω_2 and ω_3 are defined as for the static casualty assignment problem from subsection 4.1.1. The additional map ω_0 in definition (4.3) maps every casualty $\chi \in X$ to its arrival time $\omega_0(\chi) \in \mathbb{R}_{\geq 0}$ at the registration point. This is the point in time when a casualty is registered by the rescue forces as person in need of help.

The medication waiting time in hospital for a casualty is determined by actual medical treatment duration of all casualties that are assigned to the same physician and treated before. We therefore additionally define

- $\omega_4 : X \to \mathbb{R}_{\geq 0}$: Actual medical treatment duration for each casualty.

The objective for the online setting is to find an optimal mapping of casualties to physicians and vehicles for each point in time $t \in [0, T]$. Hence, the functions for the sequence of treatment have now to be defined time-dependent by

$$L_p(t) : \varphi_t^{-1}(p) \to \{1, 2, \ldots, |\varphi_t^{-1}(p)|\},$$
$$L_v(t) : \psi_t^{-1}(v) \to \{1, 2, \ldots, |\psi_t^{-1}(v)|\}.$$

Clearly, the mappings (φ_t, ψ_t) and orderings $(L_p(t))_{p\in P}$, $(L_v(t))_{v\in V}$ should be chosen, such that they minimize F_t and concurrently satisfy the condition C1 from subsection 4.1.1, again for every point in time $t \in [0, T]$.
Like for the static casualty assignment problem, the ordering sequences $(L_p(t))_{p\in P}$ and $(L_v(t))_{v\in V}$ are bijective, and the number of all possible combinations equals $\mathcal{O}(n^n)$ if n is the total number of casualties. As the static casualty assignment problem can be interpreted as a special case of the online version (all casualties arrive at a specific point in time at the disaster area), the online problem is even more complex and thus requires application of reasonable online approximation heuristics.

Regarding the notation of section 2.1, the set of solutions $\mathcal{S}$ and components C can be identified in an analogous way as for the static CASASSIGN problem. The only difference is now that the point in time is an additional part of every component. Thus, every component $c \in C$ corresponds to a 6-tuple $(\chi, v, p, L_v(t)(\chi), L_p(t)(\chi), t)$ with $\chi \in X(t)$, $v \in V$, $p \in \gamma_2(\gamma_1(\chi))$ and $t \in [0, T]$. Consequently, also the solution set is time-dependent. This means that we do not only consider one solution set $\mathcal{S}$ but a band of solution sets $(\mathcal{S}(t))_{t\in[0,T]}$, where every solution $S \in \mathcal{S}(t)$ is a collection of components, such that all casualties $\chi \in X(t)$ are assigned to an ambulance and a qualified physician with corresponding queueing positions. Again, we have $F_t = F_t(S)$, and the value of the objective function is determined by the current solution $S \in \mathcal{S}(t)$.

4.2 Implementation of GWMSA

4.2.1 Greedy

As the static and the online casualty assignment problem start from different premises, we design a separate greedy strategy for each problem variant.

4.2.1.1 Static assignment problem

The implementation of the greedy strategy for the static CASASSIGN problem is done as follows:
In every step, the set of casualties that has not yet been assigned is scanned, and an assignment consisting of a casualty, a vehicle, and a qualified physician is chosen, such that the potential increase of the damage of all already assigned casualties is

minimized. Afterwards, the assignment where this potential increase is smallest will be fixed, and the corresponding casualty will be marked as assigned. This procedure stops as soon as all casualties are assigned. Hence, the greedy algorithm for the static casualty assignment problem looks as follows:

Input:

- Set of casualties X, vehicles V, physicians P, mappings γ_1, γ_2.
- Objective function F.

Algorithm:

$X_0 := \emptyset$.
$S_0 := \emptyset$.
while $X_0 \neq X$ **do**
  $\Delta F_{min} := \infty$.
  $c_{min} := \emptyset$.
  $\chi_{next} := \emptyset$.
  for all $\chi \in X \setminus X_0$ **do**
    for all $v \in V$ **do**
      for all $p \in \gamma_2(\gamma_1(\chi))$ **do**
        Compute $\Delta F_{curr} := F(S_0 \cup (\chi, v, p, |\psi^{-1}(v)| + 1, |\varphi^{-1}(p)| + 1)) - F(S_0)$.
        if $\Delta F_{curr} < \Delta F_{min}$ **then**
          $\Delta F_{min} := \Delta F_{curr}$.
          $c_{min} := (\chi, v, p, |\psi^{-1}(v)| + 1, |\varphi^{-1}(p)| + 1)$.
          $\chi_{next} := \chi$.
        end if
      end for
    end for
  end for
  $S_0 := S_0 \cup c_{min}$.
  $X_0 := X_0 \cup \chi_{next}$.
end while

Output:

- Complete casualty assignment S_0.

Remark:

- For every casualty $\chi \in X \setminus X_0$ that has not yet been assigned to a vehicle and physician, only joinings to the end of transport and medication queues are considered when searching for a component c_{min} that minimizes ΔF. The reason is that jumping queue positions would result in a reordering of casualties in X_0 that have already been assigned, and this would imply a replacement of components that have already been added to S_0. However, this strategy, in turn, would contradict the nature of a greedy approach, where all decisions that have been taken at one point are irrevocable (revise section 3.1.1 for details).

4.2.1.2 Online assignment problem

Within the online scenario, the assignment of casualties to physicians is done after casualties have already been delivered to specific hospitals. To put it another way, this means that the hospital where a casualty is delivered to has to be fixed, before medical treatment of this casualty can be started. It is therefore independent of the decision in which order a physician in a hospital will treat casualties that are assigned to this hospital, medically. This follow-up optimization algorithm in hospitals is described next, before the actual greedy assignment strategy is introduced.

a. Follow-up optimization in hospitals

It seems reasonable to assume that processing of casualties within hospitals is also done with respect to priorities. This means that each time a physician is available, he or she has to decide which casualty of all currently waiting casualties should be treated next, such that the sum of penalties for all currently waiting casualties is minimized.
In practice, an available physician would choose a waiting casualty out of all waiting casualties, such that the sum of penalties of all other waiting casualties *after* having medicated this specific casualty is minimized. Any other more complex strategies going beyond this consideration can be assumed as impracticable not at least because of the urgent circumstances.
This leads to a follow-up optimization procedure in hospitals or, mathematically, the (final) computation of $L_p(t)$ for all $p \in P$ and each $t \in [0, T]$. This procedure, which is executed subsequently to the greedy algorithm introduced in the next paragraph, works as follows:

1. For each hospital $h \in H$ and each point in time $t \in [0, T]$:

    i. For each physician $p \in P$ that is idle at point in time t and that can medicate at least one casualty $\chi \in X$ currently waiting for medical treatment in hospital h:

- Assign this casualty χ of all waiting casualties to an available and qualified physician p (that is $\varphi(\chi) := p$), such that total damage over all other waiting casualties is minimized for the point in time when χ is medicated (based on expected medical treatment duration).

b. Online greedy algorithm

The greedy strategy for the online CASASSIGN problem processes only those casualties in priority order that are currently waiting for transport at the registration point and computes locally optimal hospital and vehicle assignments for them. Expected journey lengths to hospitals are taken into consideration. Besides, when loading ambulances with capacity greater than one, greedy assigns a casualty to the next available vehicle independent from destination hospitals of other passengers within the same cart load.
The functionality of our online greedy algorithm can be described informally as follows:

1. For each point in time $t \in [0, T]$:
   i. For each vehicle $v \in V$ that is idle at point in time t:
      I. While vehicle v has capacities left *and* there are casualties waiting for transport at point in time t:
         a. Transport this casualty $\chi \in X$ by vehicle v as next one who is waiting for transport at point in time t and suffers highest current damage $\pi_{\gamma_1(\chi)}(w(\chi))$ of all casualties currently waiting for transport.
         b. Determine destination hospital for χ:
            - Choose qualified physician $p \in P$ and feasible medication queue position for χ, such that estimated total damage value (based on expected transport and medication durations of all casualties already enregistered) is minimized. Set the location of p as destination hospital for χ.
2. Compute assignment from casualties $\chi \in X$ to physicians $p \in P$ (see above).

Remarks:

- Note that it is possible that a vehicle starts delivering casualties although it is not full yet, since the number of casualties still to come in the future is unknown.
- A medication queue position is called "feasible" if arrival time of current casualty in hospital is smaller than the point in time when the casualty at specified

position is medicated. In other words, all positions that correspond to casualties that are either still waiting for medication or have not yet arrived at hospital when current casualty arrives, are feasible.

- The assignment to a physician in step b is only tentatively, because final assignment to physicians is computed subsequently in step 2 with procedure "follow-up optimization in hospitals", see above.

4.2.2 Modified simulated annealing

As already done before and due to the different presuppositions, we define separate neighborhood relations and candidate generation probabilities for the static and the online casualty assignment problem.

4.2.2.1 Static assignment problem

a. Neighborhood definition

In the context of the static CASASSIGN problem, a solution S is an arbitrary assignment of all casualties to vehicles and qualified physicians with corresponding queueing positions. We can thus define the neighborhood of a solution $S \in \mathcal{S}$ by all solutions with the same assignments except that one casualty is assigned to a different vehicle and/or physician or another queuing position:

$$\begin{aligned} N(S) :=& \{S' \in \mathcal{S} \mid \exists \chi \in X, v, v' \in V, p, p' \in \gamma_2(\gamma_1(\chi)), i \in \{1, \ldots, |\psi^{-1}(v)|\}, \\ & i' \in \{1, \ldots, |\psi^{-1}(v')|\}, j \in \{1, \ldots, |\varphi^{-1}(p)|\}, j' \in \{1, \ldots, |\varphi^{-1}(p')|\} : \\ & S \setminus S' = \{(\chi, v, p, i, j)\} \wedge S' \setminus S = \{(\chi, v', p', i', j')\}\} \end{aligned}$$

b. Definition of modified candidate generation probabilities

The definition of modified candidate generation probabilities is based on following observations:

- A reassignation of casualties with higher damage will conjecturally have a greater effect on the global optimization function F. Therefore, we define the probability for reassigning $\chi \in X$ in a specific simulated annealing step by the ratio of its current damage $\pi_{\gamma_1(\chi)}(w(\chi))$ and the overall damage $F(S)$ for the current solution S. Remember w as the function for waiting time and $\pi_{\gamma_1(\chi)}$ as the function for the corresponding course of disease for casualty χ.

$$\mathbb{P}_X(\chi) = \frac{\pi_{\gamma_1(\chi)}(w(\chi))}{F(S)}.$$

- The probabilities for assignment of a chosen casualty χ to a specific vehicle result from similar considerations. It seems plausible that an assignment of χ to a vehicle $v \in V$ with low workload will improve the optimization. In order to estimate the workload of a specific vehicle v, we sum up the damage of all patients $\psi^{-1}(v)$ that are assigned to the specific vehicle (negative term of the numerator). By subtracting this damage from the overall damage $F(S)$, we obtain an anti-proportional dependency leading to low values if the vehicle is fully loaded. Finally, we normalize by the sum of the corresponding numerators for all vehicles.

$$\mathbb{P}_V(v) = \frac{F(S) - \sum_{\chi \in \psi^{-1}(v)} \pi_{\gamma_1(\chi)}(w(\chi))}{\sum_{\hat{v} \in V} (F(S) - \sum_{\chi \in \psi^{-1}(\hat{v})} \pi_{\gamma_1(\chi)}(w(\chi)))}. \tag{4.4}$$

- Similarly, we define probabilities for the reassignment of the chosen casualty χ to qualified doctors. Once more, an assignment of χ to a physisian p with low workload will probably improve the optimization. Let $P_\chi = \gamma_2(\gamma_1(\chi))$ be the set of physisians that are qualified to medicate the injury of χ.

$$E_\chi = \sum_{\hat{p} \in P_\chi} \sum_{\hat{\chi} \in \varphi^{-1}(\hat{p})} \pi_{\gamma_1(\chi)}(w(\hat{\chi}))$$

gives the overall damage of all casualties $\hat{x}$ that are assigned to qualified doctors for χ's injury. The probability for assigning a casualty χ to a physision p is calculated as follows:

$$\mathbb{P}_P(p \,|\, \chi) = \begin{cases} 0 & \text{if} \quad p \notin P_\chi \\ \frac{E_\chi - \sum_{\hat{\chi} \in \varphi^{-1}(p)} \pi_{\gamma_1(\chi)}(w(\hat{\chi}))}{\sum_{\hat{p} \in P_\chi} (E_\chi - \sum_{\hat{\chi} \in \varphi^{-1}(\hat{p})} \pi_{\gamma_1(\chi)}(w(\hat{\chi})))} & \text{if} \quad p \in P_\chi \end{cases}.$$

If $p \notin P_\chi$, an assignment of χ to p is not feasible, because physisian p has not the necessary qualification. If p has the qualification, the probability is high in case of a low workload for p. Once more, the negative term of the numerator specifies the damage of all casualties that are mapped to the specific physician. By subtracting this value from the overall damage of patients medicated by qualified physisians, the numerator will be higher for doctors with a low workload. The denominator normalizes the specific values by summing up the corresponding values for all qualified physisians $\hat{p} \in P_\chi$.

- To limit the complexity for generation of the next neighbor solution candidate, the transport and medication queueing positions of a reassigned casualty are not drawn workload adapted but equally distributed with respect to the length of the corresponding queues:

$$\mathbb{P}_{L1}(L_v(\chi) = i \,|\, v) = \frac{1}{|\psi^{-1}(v)|} \quad \text{for } i = 1, 2, \ldots, |\psi^{-1}(v)|,$$
$$\mathbb{P}_{L2}(L_p(\chi) = i \,|\, p) = \frac{1}{|\varphi^{-1}(p)|} \quad \text{for } i = 1, 2, \ldots, |\varphi^{-1}(p)|.$$

Let $S, S' \in \mathcal{S}$ be two solutions of the static casualty assignment problem with $S \setminus S' := \{(\chi, v, p, i, j)\}$ and $S' \setminus S := \{(\chi', v', p', i', j')\}$ for

- $\chi, \chi' \in X$, $v, v' \in V$, $p \in P_\chi$, $p' \in P_{\chi'}$.
- $i \in \{1, 2, \ldots, |\psi^{-1}(v)|\}$, $i' \in \{1, 2, \ldots, |\psi^{-1}(v')|\}$.
- $j \in \{1, 2, \ldots, |\varphi^{-1}(p)|\}$, $j' \in \{1, 2, \ldots, |\varphi^{-1}(p')|\}$.

Then altogether, the candidate generation probability is given by

$$\mathbb{P}_{cg}(S, S') := \begin{cases} \frac{\mathbb{P}_X(\chi) \cdot \mathbb{P}_V(v') \cdot \mathbb{P}_P(p' \,|\, \chi)}{|\psi^{-1}(v')| \cdot |\varphi^{-1}(p')|}, & \text{if} \quad \chi = \chi' \\ 0, & \text{else} \end{cases}.$$

4.2.2.2 Online assignment problem

In our online setting, a solution S is an arbitrary assignment of those casualties that are waiting for transport at disaster area at a specific point in time to vehicles and physicians. As already described in subsection 4.1.2, we assume groups of casualties to arrive in intermittent intervals. Each time a group of new casualties arrives, a decision about conveyance and destination hospital has to be taken for each casualty of the group. The idea is now to assign new arriving casualties tentatively to an ambulance and a qualified physician within the greedy initialization step. Afterwards, the modified simulated annealing procedure is triggered, that tries to optimize the assignment of all casualties that are currently waiting for evacuation in the optimization step. These two steps are executed at each point in time when a new casualty group arrives.

a. Neighborhood definition

Like for the static version, a neighbored solution is one with the same assignments except that one casualty is assigned to a different vehicle and/or physician or another queuing position. However, the specific point in time $t \in [0, T]$ has to be taken into account now, too:

$$
\begin{aligned}
N(S) :=& \{S' \in \mathcal{S}(t) \,|\, \exists \chi \in X, v, v' \in V, p, p' \in \gamma_2(\gamma_1(\chi)), i \in \{1, \ldots, |\psi_t^{-1}(v)|\}, \\
& i' \in \{1, \ldots, |\psi_t^{-1}(v')|\}, j \in \{1, \ldots, |\varphi_t^{-1}(p)|\}, j' \in \{1, \ldots, |\varphi_t^{-1}(p')|\} : \\
& S \setminus S' = \{(\chi, v, p, i, j, t)\} \wedge S' \setminus S = \{(\chi, v', p', i', j', t)\}\},
\end{aligned} \tag{4.5}
$$

for every $S \in \mathcal{S}(t)$.

b. Definition of modified candidate generation probabilities

The modified candidate generation probabilities for the online GWMSA algorithm are defined, based on the same observations as for the static casualty assignment problem, see subsection 4.2.2.1. The only difference is that these probabilities now additionally depend on the specific point in time $t \in [0, T]$. Consequently, for a solution $S \in \mathcal{S}(t)$ the probability for reassignation of a casualty $\chi \in X$ is given by

$$
\mathbb{P}_{X,t}(\chi) = \frac{\pi_{\gamma_1(\chi)}(w(\chi))}{F_t(S)},
$$

and for every vehicle $v \in V$ we have

$$
\mathbb{P}_{V,t}(v) = \frac{F_t(S) - \sum_{\chi \in \psi_t^{-1}(v)} \pi_{\gamma_1(\chi)}(w(\chi))}{\sum_{\hat{v} \in V}(F_t(S) - \sum_{\chi \in \psi_t^{-1}(\hat{v})} \pi_{\gamma_1(\chi)}(w(\chi)))}.
$$

The probability for assigning a casualty $\chi \in X$ to a specific physician $p \in P$ becomes

$$
\mathbb{P}_{P,t}(p \,|\, \chi) = \begin{cases} 0 & \text{if} \quad p \notin P_\chi, \\ \Theta(p, t, \chi) & \text{if} \quad p \in P_\chi \end{cases},
$$

where

$$
\Theta(p, t, \chi) := \frac{E_{\chi,t} - \sum_{\hat{\chi} \in \varphi_t^{-1}(p)} \pi_{\gamma_1(\chi)}(w(\hat{\chi}))}{\sum_{\hat{p} \in P_\chi}(E_{\chi,t} - \sum_{\hat{\chi} \in \varphi_t^{-1}(\hat{p})} \pi_{\gamma_1(\chi)}(w(\hat{\chi})))},
$$

and

$$
E_{\chi,t} = \sum_{\hat{p} \in P_\chi} \sum_{\hat{\chi} \in \varphi_t^{-1}(\hat{p})} \pi_{\gamma_1(\chi)}(w(\hat{\chi})).
$$

Here, $\hat{\chi}$ denotes all casualties that are assigned to qualified doctors but not yet under medical treatment at point in time $t \in [0, T]$. Note that for the previous considerations about assignment probabilities at each point in time only those casualties were taken into account that have already arrived at disaster area. The reason is that we assume the type and final number of casualties still to come in the future to be unknown. Furthermore, for computing ambulance and physician assignment

probabilities, those casualties that have already been transported to destination hospital or have already been treated medically, respectively, are omitted, since their assignment is irrevocable. The same condition holds for the choice of the queueing positions, which are again chosen equally distributed to limit complexity of neighbor solution candidate generation:

$$\mathbb{P}_{L1,t}(L_v(t)(\chi) = i \,|\, v) = \frac{1}{|\psi_t^{-1}(v)|} \quad \text{for } i = 1, 2, \ldots, |\psi_t^{-1}(v)|,$$
$$\mathbb{P}_{L2,t}(L_p(t)(\chi) = i \,|\, p) = \frac{1}{|\varphi_t^{-1}(p)|} \quad \text{for } i = 1, 2, \ldots, |\varphi_t^{-1}(p)|.$$

Besides, we want to point out that all computations of casualty damages are based on *expected* journey lengths to destination hospitals and medical treatment durations, since the actual values are only known *after* evacuation and medical treatment of a casualty has already started (and thus, when assignment is already irrevocable).

The candidate generation probabilities are the same as for the static casualty assignment problem but now additionally also depend on the specific point in time $t \in [0, T]$:

$$\mathbb{P}_{cg}(S, S') := \begin{cases} \frac{\mathbb{P}_{X,t}(\chi) \cdot \mathbb{P}_{V,t}(v') \cdot \mathbb{P}_{P,t}(p' \,|\, \chi)}{|\psi_t^{-1}(v')| \cdot |\varphi_t^{-1}(p')|}, & \text{if} \quad \chi = \chi' \\ 0, & \text{else} \end{cases} \tag{4.6}$$

for $S, S' \in \mathcal{S}(t)$ with $S \setminus S' := \{(\chi, v, p, i, j, t)\}$ and $S' \setminus S := \{(\chi', v', p', i', j', t)\}$.

c. Execution times

Like for the other problem classes, the modified simulated annealing algorithm for the online casualty assignment problem is given by the abstract algorithm specification from subsection 3.1.2.3 when replacing the abstract definitions for neighborhood $N(S)$ and candidate generation probabilities $\mathbb{P}_{cg}(S, S')$ by the concrete definitions (4.5) and (4.6) from above. However, this time the algorithm from subsection 3.1.2.3 is not only executed once but every time a new casualty group arrives. So, if a new casualty group arrives at disaster area, the greedy algorithm from subsection 4.2.1.2 is used to compute an initial assignment for the recently arrived casualties first. Afterwards, the modified simulated annealing procedure is triggered to optimize this initial assignment. At this, casualties that are still waiting for transport at disaster area or medication in hospital are included within the optimization procedure. Casualties that are already under medical treatment are not considered, as their assignment is irrevocable.

4.3 Empirical simulation

4.3.1 Static assignment problem

We have evaluated the performance of the GWMSA implementation for the static CASASSIGN problem by simulating a practical example in the context of a football match at the arena of Frankfurt (Main) during the FIFA soccer world cup 2006 in Germany. For the given example, we first compare GWMSA with a single greedy assignment. Afterwards, we discuss how the different initialization strategies (random versus greedy) and candidate generation probabilities (uniform distribution versus workload adapted) affect the performance of simulated annealing.
Before explaining our studies in detail, we will first derive appropriate penalty functions that describe the course of an injury and introduce the data sets.

4.3.1.1 Penalty functions

Overall, we define 23 different sample functions for injuries such as luxation, craniocerebral injury, bleeding, or serious dermal burn. These differ both in their initial priority ($t = 0$) and in the positive functional course (e.g., $\pi(t) = \ln(t+1)$, $\pi(t) = t^2$, $\pi(t) = e^t - 1$, $\pi(t) = \sqrt{t}$). Clearly, a longer waiting time causes a higher penalty value for all functions. For later evaluation, we assign those 23 functions to the triage groups as defined in subsection 4.1.1.2 considering the value for $t = 0$ similar to initial priorities. 11 out of 23 functions are assigned to the first triage group T1, 6 to the second group T2 and the remaining 6 to the third group T3. We do not consider T4, since those subjects do not have an impact on the overall result. We once more point out that an introduction of such functions is useful and feasible, after consultation with experts. The chosen functions are, however, elected exemplary to show the power of this approach and need to be adapted by experts in realistic future scenarios.

4.3.1.2 Data setup

This subsection introduces the training and test data that will be used to show the efficiency of GWMSA for optimization of emergency supply for mass events. The factors affecting the optimization problem are manifold. Following values have to be specified both for training and for testing the algorithm: Number of casualties and corresponding injuries, duration for medication for each casualty, number of hospitals including number and type of physicians, number of vehicles, and a matrix specifying the transportation time between the place of catastrophe and the hospitals. Since our GWMSA implementation for the static CASASSIGN problem should

work in arbitrary settings, the training will consider artificial data that is randomly drawn. For testing, we evaluate a disaster during a football game in Frankfurt.

a. Training data

For training the parameters of simulated annealing, we have generated 50 instances of a random training set as follows: For each of the factors mentioned above (e.g., number of casualties, injuries) we randomly draw a value from a uniform distribution within reasonable intervals, see table 4.1. Because the training set is generated randomly and has a representative amount of training instances, the method should generalize to many problems in this domain. A consideration of all combinations over the factors is not realistic. The number of degrees of freedom is certainly too high. Later, we will see that a higher number of instances will not lead to improved results. Thus, a set of 50 different training instances is sufficient.

variable	**left bound**	**right bound**
number of casualties	200	1600
medication duration	5 minutes	2 hours
initial priority	10	1000
number of hospitals	3	20
number of physicians	50	500
number of vehicles	5	25
vehicle capacity	1	4
journey length to hospital	5 minutes	40 minutes

Table 4.1 Interval bounds for randomly generated input variables in the training set.

b. Test data

For testing, we use the arena disaster scenario at the football world cup 2006 as described in [45]. The number of casualties is fixed at 1000. The distribution between different triage groups (T1 to T4) has been assumed as 20%-40%-20%-20% as illustrated in figure 4.4. This leads to 400 casualties assigned to triage group T2 and 200 casualties to all remaining groups. Altogether, 800 casualties (ignore T4) with a significant chance for viability have to be served and considered in the optimization problem. We will later explore the affiliation between the 23 injuries and the triage groups. We are thus able to draw injuries for the casualties for each group randomly, leading to 800 casualties distributed to T1-T3 as specified by [45].

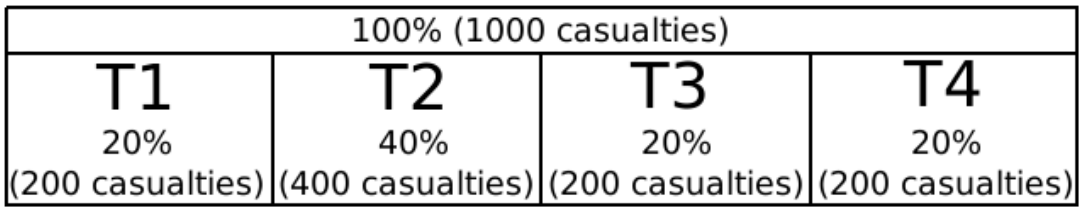


Figure 4.4 Distribution of 1000 casualties to triage groups.

To obtain realistic values for the number of hospitals including the number and type of physicians as well as for the number of available vehicles, we have collected data from 20 quality reports from different hospitals [30] around Frankfurt. Finally, we estimate the transportation time matrix between the arena and the hospitals using a route planning tool. The medication duration for the randomly generated casualty test set is chosen in the same range as for the training set (between 5 minutes and 2 hours).

4.3.1.3 Benchmark strategies

In the context of our empirical evaluation of GWMSA applied on the static casualty assignment problem we will discuss the effect of two kinds of initializations (random and greedy) and two ways for generating neighbored solutions (equally distributed and modified). We will compare the performance of the following configurations of simulated annealing:

1. Random initialization/Random candidate generation (classic simulated annealing, “CSA”).
2. Greedy initialization/Workload-aware candidate generation (“GWMSA”).
3. Random initialization/Workload-aware candidate generation (“Hybrid I”).
4. Greedy initialization/Random candidate generation (“Hybrid II”).

4.3.1.4 Simulation results

Subsequently, we make two different experiments to evaluate the presented methods. The first experiment compares the performance of a single application of greedy with that of its combination with modified simulated annealing (GWMSA). The second experiment discusses how the different initialization strategies and candidate generation probabilities influence the performance of simulated annealing.

a. Training

In order to find adequate parameters for our scenario, we continuously generate 1000 different parameter triples randomly composed of initial temperature values, number of iterations, and temperature degression coefficient. Then, we calculate for each triple and each instance of the training data from paragraph a of subsection 4.3.1.2 an assignment of casualties to vehicles and physicians with the GWMSA algorithm. The triple leading to the minimum standardized average damage is saved and used later in the test. Note that results for different numbers of casualties cannot directly be compared. Therefore, we normalize with respect to the number of casualties. We will show that 100 triples are already sufficient, because a consideration of up to 1000 does not significantly improve the results. The procedure of generating and evaluating new triples is stopped as soon as no significant improvement can be observed. The results of this training procedure will be discussed in detail in paragraphs c and d.

b. Testing

In the test phase, we apply GWMSA to the test data from paragraph b of subsection 4.3.1.2 using the parameter triple trained in the previous paragraph.

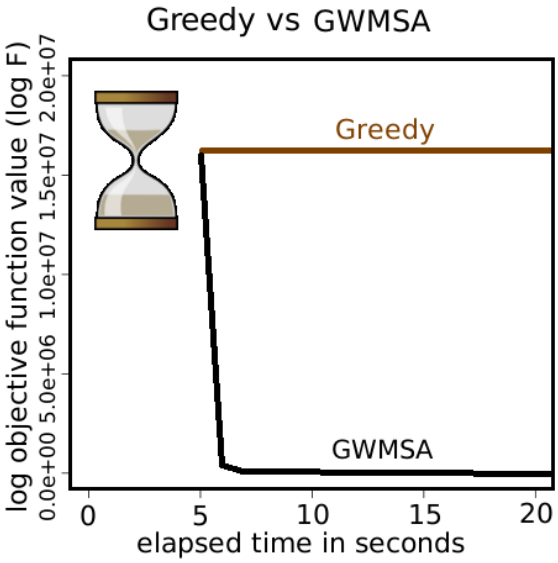


Figure 4.5 Overall damage over time when applying greedy and GWMSA on test set.

As a result, the greedy solution can be improved significantly by an additional optimization with modified simulated annealing. Figure 4.5 illustrates the performance (with respect to the total damage F) of both algorithms over time (mind the

logarithmic scale of the F-axis). Greedy elapses after 5 seconds. Already after additional 1-2 seconds of computation, modified simulated annealing converges thereby decreasing the optimization value significantly by over 99% in comparison to greedy (compare values in Table 4.2).

technique	F finally	elapsed time
Greedy	16225142.251736	5 seconds
GWMSA	53306.985539	7 seconds

Table 4.2 Optimization results when applying greedy and GWMSA on the test data set.

The main reason for this result is the fact that greedy only minimizes with respect to the increments of current damage of all casualties that have already been assigned. Impacts on the global damage cannot be taken into account as this value is unknown yet when greedy has to make a decision which casualty should be assigned next. Modified simulated annealing, in contrast, changes assignments of casualties considering global damage contribution of their current assignments and is thus able to decrease global damage additionally. Although the computation time is higher, GWMSA achieves excellent results in less than 10 seconds (passable time frame for scheduling). As shown later, the good performance results mainly from the sophisticated generation of neighbored solutions with modified candidate generation probabilities.

c. Discussion of simulated annealing

The following discussion of simulated annealing is twofold. First, we will compare the four variants of simulated annealing (CSA, GWMSA, Hybrid I, Hybrid II) for assignment of casualties to vehicles and physicians. Second, we will discuss how the number of parameter triples during training and the number of training instances influence the performance of simulated annealing.
To evaluate performance of the four simulated annealing configurations in our scenario, we first apply them on the training instances. The results are shown in Figure 4.6. The average damage with respect to number of training instances is displayed on the ordinate axis.
On average, GWMSA performs best on all training instances, as indicated by the low average damage. One can derive from three larger steps in the red curve that a random initialization in Hybrid I results in very bad optimization value for three training cases. Note that for the corresponding instances, the algorithm gets stuck in a bad local minimum leading to a high damage thereby escalating the average damage (cf. figure 4.6). The same issue also applies to CSA due to a random initialization. Hybrid II is both error-prone to difficult training instances and additionally

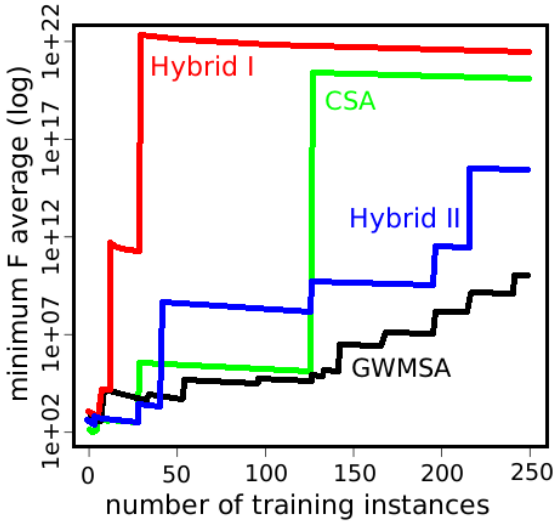


Figure 4.6 Average damage F for varying number of training instances and fixed optimal parameters.

achieves worse average results than GWMSA. An evaluation of these difficult instances shows that they are caused by a full workload of a specific doctor without alternatives. This fact clearly leads to high overall damage. Although such cases are exceptional they are nevertheless important and have to be taken into account. A visual inspection of the four curves emphasizes that GWMSA is much more robust (lower leaps). Since we aim for a method that is general enough to apply to as many settings as possible (and an unbalanced distribution of injured to doctors is realistic for disasters), GWMSA is the most robust choice. This choice is reinforced by the algorithms' runtime. Table 4.3 summarizes the average computation times for all four configurations.

technique	average computation time
CSA	34.4113 seconds
GWMSA	10.0409 seconds
Hybrid I	13.5718 seconds
Hybrid II	26.0926 seconds

Table 4.3 Average optimization time till convergence on training set for all four configurations.

Obviously, the configurations considering the sophisticated and workload-aware search strategy converge much faster in less than half the time. Because of the greedy initialization, GWMSA is even 3 seconds faster than Hybrid I on average. It seems that GWMSA, caused by the greedy initialization and the sophisticated generation of neighbored solutions with modified candidate generation probabilities, both starts closer to the optimum and converges faster.
Initialization by greedy takes 2.3 seconds on average while a random initialization is computed in less than one second (see table 4.4).

technique	**average computation time**
Random initialization	0.215 seconds
Greedy initialization	2.34965 seconds

Table 4.4 Average computation times for random and greedy initialization.

Due to the virtues of a greedy initialization in combination with a workload-aware searching strategy a total computation time of less than 15 seconds is realistic and sufficient. The results were additionally validated on the test set (see table 4.5).

technique	**F after init**	**F finally**	**iterations**	**elapsed time**
CSA	$1.8779 \cdot 10^{18}$	187370.598831	15080	3 seconds
GWMSA	16225142.25	53306.985539	183551	40 seconds
Hybrid I	$1.8779 \cdot 10^{18}$	52913.271735	165388	37 seconds
Hybrid II	16225142.25	61652.378285	237088	40 seconds

Table 4.5 Optimization results on test set for all four configurations of simulated annealing

The overall damage after initialization is best for GWMSA and Hybrid II, as one can expect because of the greedy initialization. The algorithms considering the modified candidate generation probabilities significantly outperform the remaining. Additionally, the runtime till convergence is lower for methods considering the modified candidate generation probabilities (CSA is not considered because of poor results). Although Hybrid I performs slightly better than GWMSA, both for duration and the optimization result on that specific test instance, we have shown before that GWMSA is much more robust and therefore first choice.

d. Important factors while training simulated annealing

In this paragraph we investigate the changes of the average damage when increasing the number of parameter triples and training instances. Therefore, we have chosen

50 different training instances and 1000 parameter triples in total for training of simulated annealing so far. Figure 4.7 (left) shows the performance measured by the average damage of GWMSA with respect to the number of generated parameter triples. We can see that the average damage converges quickly already after 10 generated parameter triples. Hence, we just need to test a few different parameter triples to find a good one that performs well on all 50 training instances.

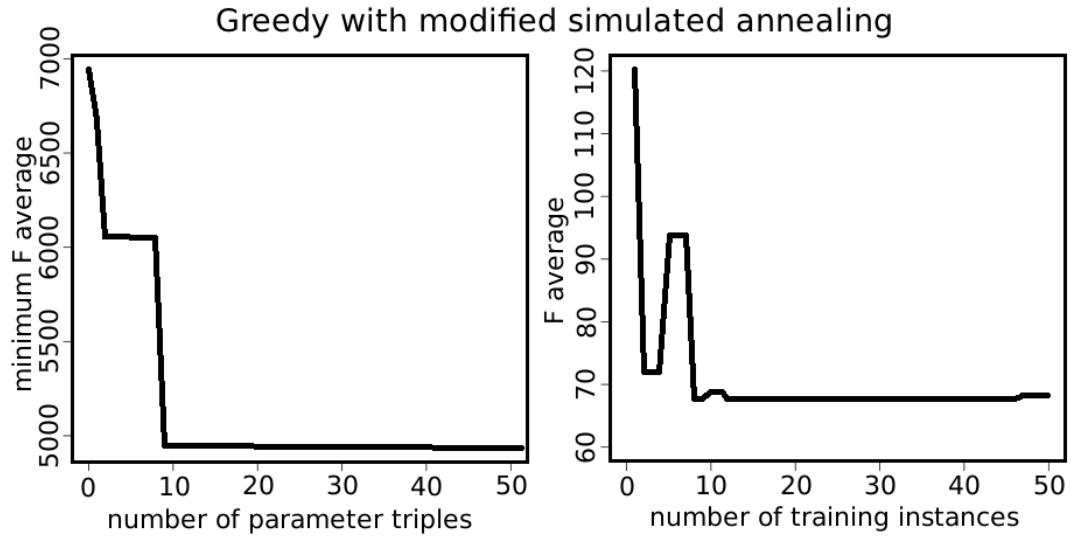


Figure 4.7 Minimum average damage F for varying number of parameter triples and fixed number of training instances (left) and average damage F for the optimal parameter triple when applied on a varying number of training instances (right).

Figure 4.7 (right) shows the average damage on the test set with respect to number of randomly generated training instances. For each number of generated training instances the currently optimal parameter triple is applied on the test set, and the resulting damage is displayed on the ordinate axis in figure 4.7 (right). Similar to figure 4.7 (left) we can see only minor changes of the average damage for more than 10 instances. Thus, 50 training instances are sufficient as representative set for finding a parameter triple that performs well on an arbitrary test instance.

e. Conclusion

We have seen that using the GWMSA strategy speeds up convergence and leads to better results of up to 99% compared to greedy when applied on instances of the static casualty assignment problem. Moreover, initialization using greedy clearly outperforms a random initialization, because with greedy the simulated annealing iterations start from an initial state that is much more close to the optimum than for

random initialization. Finally, we have evaluated the performance of simulated annealing in different settings. The combination of a greedy initialization and modified candidate generation probabilities turns out to be the most robust setting on many training instances, whereas other settings that involve random initialization and/or equally distributed candidate generation probabilities perform worse especially on "extreme" training instances.
One issue for future research would be to try to find more realistic penalty functions in cooperation with medical expert staff. Furthermore, other training strategies and optimization techniques like genetic approaches should be investigated.

4.3.2 Online assignment problem

In our experimental studies regarding the online CASASSIGN problem we will compare three different strategies for online assignment of casualties to available ambulances and physicians in hospitals. In particular these are the D'Hondt assignment strategy, which is used in today's practice, our greedy strategy from subsection 4.2.1.2, and the complete implementation of the GWMSA strategy using modified simulated annealing from subsection 4.2.2.2. As already done for the static casualty assignment problem in last subsection, we will investigate the additional benefit when applying modified simulated annealing iterations subsequently to greedy for the online assignment problem. The comparison of the three strategies will be based on a relaxed version of our assignment problem, which provides a lower bound of the optimal solution and can be efficiently computed.

4.3.2.1 Data setup

As already done for the static casualty assignment problem we will use empirical as well as randomly generated data samples for our simulations.

a. Empirical data

For testing, we use the same arena disaster scenario at the football world cup 2006 as for the static casualty assignment problem, see paragraph b of subsection 4.3.1.2. We also adopt the distribution over triage groups from figure 4.4.

b. Random data samples

For performance evaluation and comparison, this time we have randomly generated 1000 instances of random data sets within the interval bounds given in table 4.1.
In our online scenario, injured persons that need medical supply are registered subsequently in groups by medical field help. Thus, decisions about further treatment of casualties (for example in which hospital a casualty should be moved) have to be

made seperately for each group not knowing at this point in time if and how many casualties are still to come.
For testing the GWMSA algorithm and the benchmark strategies described in the next subsection, we therefore model arriving casualty groups as a Poisson process with variable jump size (which corresponds to the size of a casualty group). The jump size is drawn randomly with an equal distribution between 1 and 10.

Given a (fixed) instance (set of casualties, hospitals, vehicles et cetera) drawn from intervals given in table 4.1, the actual journey length to hospital and the actual medical treatment duration of casualties are drawn normally distributed around their expected values with standard deviations. The assumption of randomly biased medical treatment durations and journey lengths is reasonable, since in practice usually there also exist exogenous environment variables that influence these quantities like for example current traffic situation or possible complications during a surgery. Consequently, these actual values are unknown when an assignment decision for a casualty has to be taken. They are drawn *after* a casualty has arrived at destination hospital or has already been medically treated, respectively. The optimization procedure is therefore done under uncertainty and based on the corresponding expected values.

4.3.2.2 Benchmark strategies

In this subsection we introduce two further benchmark strategies to evaluate the performance of our GWMSA implementation. The first one is the D'Hondt algorithm, which is currently used in practice to compute an online assignment of waiting casualties to hospitals, and the second one is a brute force strategy that computes the optimal solution of a relaxed optimization problem. This relaxation result will be used later as a lower bound for the solution of the original problem instances.

a. D'Hondt

The city counsil of Frankfurt has arranged with each regionally hospital in negotiations how many casualties are taken in by this hospital if a mass casualty incident occurs. To ensure a balanced assignment of casualties to hospitals that regards the negotiated accommodation numbers, they make use of the D'Hondt method for online assignment.
The D'Hondt method is a highest averages method for allocating seats in party-list proportional representation. The method is named after the Belgian mathematician Victor D'Hondt. This system, which is used by many legislatures all over the world, slightly favors large parties and coalitions over scattered small parties, see [115].
Furthermore, when loading ambulances with casualties usually the attempt is made

to assign all casualties with the same destination hospital to one ambulance (of course, this is only relevant for ambulances with load capacity greater than one). Practically this means that each time a casualty is assigned to a vehicle that was empty before, the queue of waiting casualties is scanned for other casualties with the same destination hospital.
Translated to our online assignment problem the corresponding algorithm looks as follows:

1. For each arriving casualty group:
   i. Assign arriving group to hospitals with respect to D'Hondt strategy.
2. For each point in time $t \in [0, T]$:
   i. For each vehicle $v \in V$ that is idle at point in time t:
      I. Transport this casualty $\chi \in X$ by vehicle v as next one who is waiting for transport at point in time t and suffers highest current damage $\pi_{\gamma_1(\chi)}(w(\chi))$ of all casualties currently waiting for transport.
      II. Search for waiting casualties that are assigned to the same hospital as casualty χ from previous step and also assign them to available vehicle v (only if vehicle has capacities left and if there exist such casualties).
3. Compute assignment from casualties $\chi \in X$ to physicians $p \in P$.

The assignment of casualties to physicians (step 3) is done by the procedure "follow-up optimization in hospitals" described in paragraph a of subsection 4.2.1.2.

b. Relaxations

In this paragraph we will define a relaxed version of our optimization problem which will serve as standard of comparison for all online optimization strategies. In this relaxed version we completely abstract from any waiting times for transport and medication. This means that the number of vehicles and the number of physicians of a specific type in a hospital is unlimited. As a consequence, each casualty can be transported immediately by a vehicle and medically treated by a physician as soon as the affected person has arrived at some hospital where the specific type of injury can be medicated. It follows that the sequences $(L_p)_{p \in P}$ and $(L_v)_{v \in V}$ in which casualties are processed are irrelevant, and it is an optimal strategy to transport each casualty to the nearest hospital where he or she can be treated medically.
The considerations from above yield to a lower bound for the minimal (optimal) value of our optimization function F with respect to the original problem definition.

This lower bound can be computed in $\mathcal{O}(|X| \cdot |P|)$, where $|X|$ is the total number of casualties, and $|P|$ is the number of physicians.

4.3.2.3 Simulation results

In this subsection we will compare the performance of the algorithms for the online casualty assignment problem. Namely these are the D'Hondt assignment strategy, our greedy algorithm and GWMSA. The performance evaluation is done with respect to the relaxation introduced in the last paragraph. As done for the static casualty assignment problem, firstly we try to find a parameter triple composed of initial temperature value, number of iterations, and temperature degression coefficient for modified simulated annealing that performs well on all instances in a training step.

a. Training

The general procedure of the training step is the same as for the static casualty assignment problem (see paragraph a of subsection 4.3.1.4 for details). The performance of 100 input parameter triples on 100 randomly generated problem instances was compared, and the one parameter triple was chosen where average damage of each casualty was smallest. In doing so, the following parameter combination turned out to show the best performance and will be used for all further computations of modified simulated annealing:

- Initial temperature: $T_{init} = 75$.
- Number of runs with constant temperature: $\xi = 82$.
- Temperature degression coefficient: $\delta = 0.02$.

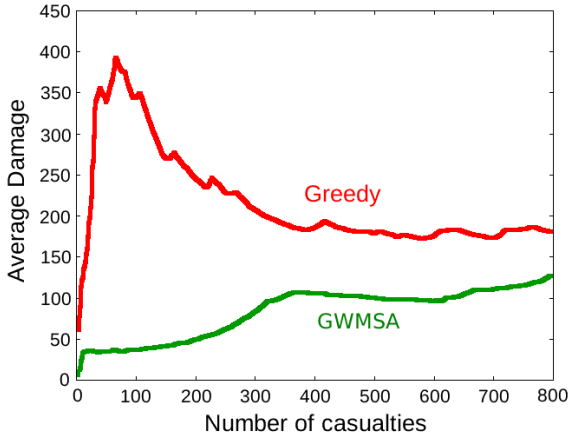
b. Test

Our tests consist of two parts. Firstly we want to evaluate the performance of our online algorithms on the empirical data set from paragraph b in subsection 4.3.1.2. In a second step we will show that these results can be generalized for arbitrary data sets that are representative for our problem definition.

c. Empirical evaluation

The final values of the average damage per casualty for the empirical data set are displayed in figure 4.8 (right). We can see that the average damage is about 30% smaller for GWMSA than for a single application of greedy. The final value for D'Hondt is with 966.24 multiple times larger than for the other strategies.
The underperformance of the D'Hondt strategy can easily be explained with the

fact that it does not take journey length to hospitals into account which, in turn, has a significant influence on casualty damage.
The evolution of average damage per casualty for GWMSA and greedy is plotted in figure 4.8 (left) (we have omitted the illustration of D'Hondt, since this strategy does not seem able to compete on the test set). On the abscissa the number of casualties that have already been treated medically is displayed, and the level of average damage these casualties suffer from is the meaning of the ordinate values.



technique	average F	lbq
D'Hondt	966.24	21.33
Greedy	181.24	4.0
GWMSA	129.05	2.85
LB	45.28	1.00

(LB = Lower bound from relaxation, lbq = lower bound quotient)

Figure 4.8 Evolution of average casualty damage on test set for Greedy and GWMSA (left) and final results for all techniques (right).

We can see that at the beginning, average damage for greedy increases rapidly and reaches its peak when having medicated almost 100 casualties. Each of them suffers a damage value of 400 on average at this point in time. Afterwards, the average damage decreases and levels off at 180 after having treated 400 casualties medically. The damage progress for GWMSA is clearly more smooth. After a rapid increase of average damage for the first 10 casualties the graph is just sloping upwards slightly up to a level of 100 after having medicated about 400 casualties and stagnates subsequently. Finally, the average damage slightly increases again a little bit for the last 200 casualties up to a level of 130.
Altogether we can conclude that the assignments computed by GWMSA are much more steady than for greedy with respect to the number of medicated casualties. Especially the casualties medicated first within the greedy strategy obviously suffer high damage. This progress could be connected to the natural course of an incident: The first casualties that are enregistered can be transported and medically treated immediately, since available capacities are still in good supply. As more casualties are enregistered, bottlenecks in connection with transport evolve and congestions in hospitals occur. Particularly in this phase the choice of a well suited process order of casualties is crucial. Apparently at this, GWMSA profits from its more detailed solution space examination in comparison to greedy when deriving a good casualty

transport and medication process order. After some time, when more casualties have been supplied, the situation relaxes.

d. Generalization of empirical results

To be able to generalize the empirical test set results from above, we have generated 1000 additional random data samples (different from those used in the training step). We applied each of the algorithms on every data set and computed the average damage with respect to the number of instances and casualties. Furthermore, we have computed the quotient of average casualty damage derived by a technique and the lower bound for average casualty damage resulting from the relaxed problem definition. The results are shown in table 4.6.

technique	**average F**	**lbq**
D'Hondt	8673.57	17.15
Greedy	1332.68	2.66168
GWMSA	1203.13	2.39815

Table 4.6 Optimization results for randomly generated data samples

The second column contains the average damage each casualty suffers from. Like already detected for the empirical test set, the results show a clear outperformance of greedy in comparison to the D'Hondt assignment strategy. The final results for both strategies were available after about one second of computation time for each instance (all computations in this work were executed on an AMD Opteron Processor with 2.60 GHz and 8 GB of RAM).
We can furthermore see that the average result improvement for GWMSA in comparison to greedy is smaller for the general case than for our empirical test set (compare results in table of figure 4.8 and table 4.6). However, the average damage suffered by each casualty can still be leveled down by about 10% when applying modified simulated annealing iterations subsequently to greedy. The average result improvement of 10% for the general case when applying GWMSA is also much smaller than for the static casualty assignment problem, where this strategy brought 99% improvement compared to greedy results (review paragraph b in subsection 4.3.1.4 for details). This fact can be explained with the smaller size of the solution space in the online scenario compared to the static problem definition which can be seen as a special case of the online assignment problem (all casualties are enregistered at the same time). Due to temporal dependencies, the number of feasible assignments is less than than for the static problem, and thus, the greedy result is "less suboptimal" for the online problem. Since additional computational effort for execution of

modified simulated annealing is negligible (just about one second for each arriving casualty group on average) it is nevertheless the strategy of choice for the online assignment problem as already 10% result improvement can have a vital effect in the context of a mass casualty incident.

e. Summary

We have seen that a simple greedy strategy clearly outperforms an assignment based on the D'Hondt algorithm which is, best to our knowledge, the only technique currently used in practice to deal with the online casualty assignment problem. It appeared that the average damage value for each casualty (and so also total damage) yielded by the greedy strategy can be additionally reduced by about 10% with an acceptable computational expenditure when applying modified simulated annealing subsequently by using the greedy result as initial solution. Thus, altogether we can conclude that the GWMSA strategy also works well for the given online assignment problem.

5 Extended test suite reduction

5.1 Problem specification

5.1.1 Introduction

Testing software functionality is a major task within the software engineering process. A software program usually consists of a set of functions (or procedures, methods, etc.). To ensure that the software is working correctly, each individual function has to be tested, which means that every function has to be called at least once by an appropriate collection of test programs.
In practice, this is usually done by compiling a test suite, which consists of many test cases. Each test case, in turn, is a sequence of function calls and can thus also be interpreted as a test program. As already mentioned, at the end, every function of a software has to be covered (called) by at least one test case of a test suite to ensure complete validity. Given this requirement it is clear that the complexity of a test suite should be limited, since testing is time-consuming and occasions costs. The complexity of a test suite is measured by the total number of function calls summarized over all test cases the suite contains, and hence, the goal is to minimize this number. Descriptively, each function should be called at least once in a test suite but also not more often than necessary. This procedure, which is also known as test suite reduction, is equivalent to solving the set covering problem (see section 3.2) and is thus $\mathcal{NP}$-complete.
However, industrial experience has shown that another objective is to balance out the number of function calls over all functions. This can be described as follows: Suppose we are given two functions e_1 and e_2 of a software program. Furthermore, let S_0 and S_1 be two test suites with identical total number of function calls, so that they are equal with respect to the first objective. Let us assume that e_1 is called once, and e_2 is called five times in S_0, whereas e_1 and e_2 both are called three times each in S_1. In this case it is a practical requirement that S_1 should be preferred over S_0, as it makes the coverage of e_1 more reliable. We will discuss this balancing requirement more detailed in subsection 5.1.2.2.
We will take the balancing requirement into consideration by additionally minimizing the variance of function calls, which measures the average squared difference between individual and average number of function calls. As test cases are sequences of function calls, we will also show how this variance concept can be generalized to higher dimensions by regarding not only the variance of single function calls but also of subsequences of function calls.

All these considerations result in a new optimization problem that we will be described in the next subsection.

5.1.2 Formalization

We will now formulate a new optimization problem that takes the two objectives mentioned above into account, namely minimizing the total number of function calls and balancing their distribution. We will start with the simple test suite reduction problem and show its equivalence to the set covering problem. The balancing requirement will be incorporated afterwards.

5.1.2.1 Equivalence to set cover

Using the definitions from section 3.2.1, C corresponds to an unreduced test suite, and every $c \in C$ corresponds to a test case. $\mathcal{U}$ contains the list of functions under test, and for every $c \in C$ the number of function calls is given by $\lambda(c)$. For every test case $c \in C$ the binary variable x_c defines whether it is contained in a reduced test suite ($x_c = 1$) or not ($x_c = 0$).
So, the task is to find a subset $S \subset C$ of test cases, such that the total number of function calls (given by (3.4)) is minimized, and every function is called at least once by some test case $c \in S$ (given by inequalities (3.5)), coincidently.
Obviously, with respect to our abstract problem definition from subsection 2.1, the set of components C can be identified with the original test suite, and every component $c \in C$ corresponds to a test case. Consequently, the set of solutions $\mathcal{S}$ contains all those collections of test cases $S \subset C$ where every function is covered.

5.1.2.2 Incorporation of balancing requirement

Before we formalize the balancing requirement we want to explain more detailed why taking it into consideration is reasonable.
Let us start with the simple example depicted in figure 5.1 together with two test suites

$$S_1 = \{(A, C, D), (A, C, E), (\mathbf{A}, C, F), (B, C, G)\}$$

and

$$S_2 = \{(A, C, D), (A, C, E), (\mathbf{B}, C, F), (B, C, G)\}.$$

Both test suites S_1 and S_2 of the example are minimal with respect to the number of function calls and vary only in one function call, which is denoted in bold font. Functions A and B are used an equal number of times in S_2, while they are not in S_1.

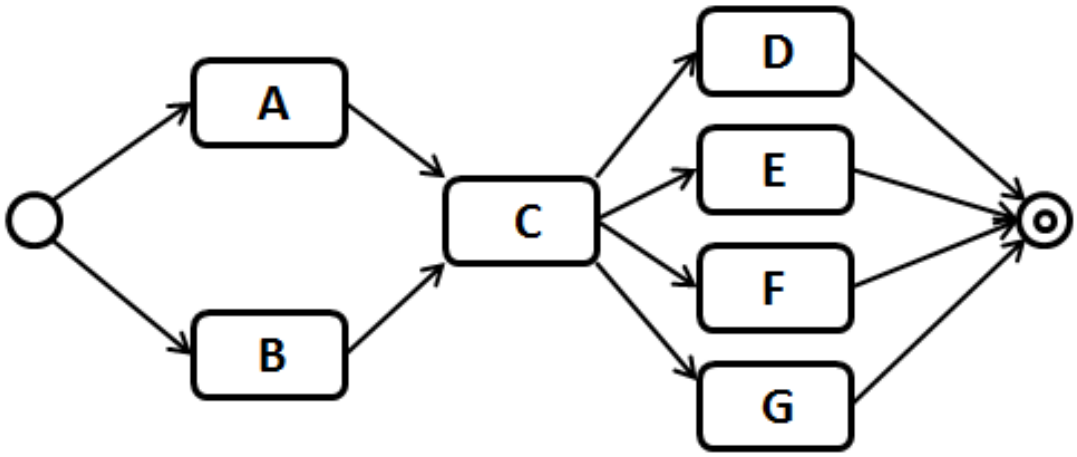


Figure 5.1 Test model example 1.

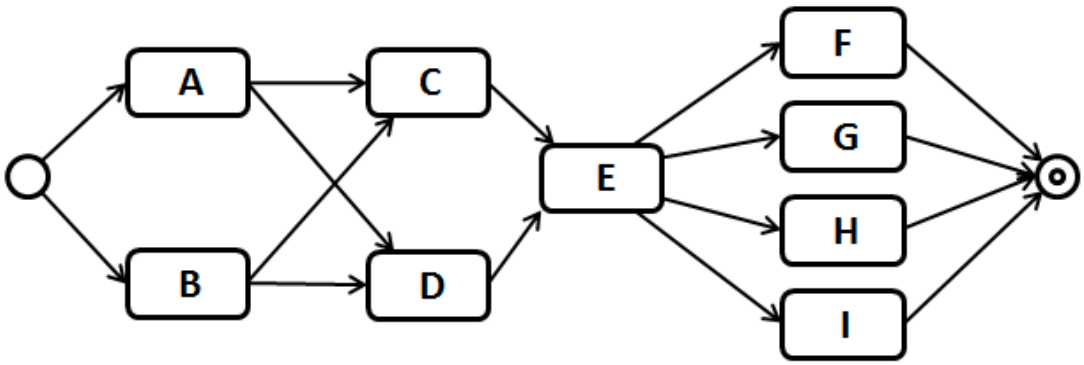


Figure 5.2 Test model example 2.

From an industrial experience, the second test suite is more desirable as it increases confidence of model based test users. Therefore, a *smoother test case distribution* or a better *distribution quality* is another practical requirement.
Let us now move on to the more complex example depicted in figure 5.2, which again comes with two test suites

$$S_3 = \{(A, C, E, F), (A, \mathbf{C}, E, G), (B, D, E, H), (B, \mathbf{D}, E, I)\}$$

and

$$S_4 = \{(A, C, E, F), (A, \mathbf{D}, E, G), (B, D, E, H), (B, \mathbf{C}, E, I)\}.$$

Both test suites are minimal, and their difference is marked in bold font. However, this time all functions are called the same number of times in S_3 as in S_4. Nevertheless, the second suite can be regarded as having a smoother test case distribution, since it includes four different variations for the first two functions while the first one only includes two. In a similar fashion, one could obtain larger examples where

distribution quality only depends on the number of occurrences of even larger test case subsequences.

5.1.2.3 Definition of function call variance

The natural quantification of variations of a specific factor is generally given by the classic statistical variance. Assigned to the current problem with definitions from sections 3.2.1 and 3.2.2, we can define the variance of function calls for a given reduced test suite S as follows:
Using (3.8), the *mean coverage* over all functions in $\mathcal{U}$ is given by

$$\overline{\kappa(\mathcal{U}, S)} := \frac{1}{|\mathcal{U}|} \cdot \sum_{e \in \mathcal{U}} \kappa(e, S),$$

where $|\mathcal{U}|$ is the total number of functions. Then, we obtain the variance as

$$\mathrm{Var}(\mathcal{U}, S) := \sum_{e \in \mathcal{U}} \frac{(\kappa(e, S) - \overline{\kappa(\mathcal{U}, S)})^2}{|\mathcal{U}|}. \tag{5.1}$$

5.1.2.4 Extended optimization problem

To formalize the problem of test suite reduction with balancing requirement, the two objectives, namely minizing the total number of function calls expressed by (3.6) and minimizing the variance (5.1), have to be linked together algebraically. Note that the variance can become zero, for example if a collection of test cases $S \subset C$ is available, such that every function $e \in \mathcal{U}$ is called exactly once (that is $\kappa(e, S) = 1$ for every $e \in \mathcal{U}$). Hence, a multiplicative conjunction of the two objectives is problematic. We will therefore add (5.1) to (3.6) and obtain a new objective function

$$G(S) := \sum_{c \in S} \lambda(c) + \sum_{e \in \mathcal{U}} \frac{(\kappa(e, S) - \overline{\kappa(\mathcal{U}, S)})^2}{|\mathcal{U}|}. \tag{5.2}$$

The side conditions (3.7) can be adopted unchanged, and by plugging in the definition of κ they are equivalent to

$$\kappa(e, S) \geq 1 \quad \forall\, e \in \mathcal{U}. \tag{5.3}$$

5.1.2.5 Variances of higher order

Both test suites S_3 and S_4 from the second example in figure 5.2 are optimal with respect to function (5.2). To be able to distinguish between them, we suggest to

generalize the coverage concept from above to tuples of function calls as follows:
In all preceding considerations the ordering of function calls within a test case $c \in C$ was irrelevant as $\lambda(c)$ and the variance (5.1) are invariant under reordering of those. However, the functions within a test case are actually always called in a predetermined order, and for the following definitions the ordering of function calls will indeed be relevant. From now on we will therefore identify all test cases $c \in C$ not as *sets* of function calls but as *finite sequences* of them. This means each $c \in C$ is of the form

$$c = (e_{c_1}, e_{c_2}, \ldots, e_{c_{|c|}}).$$

Now, let $\mathcal{U}_C^k$ be the set of all function call subsequences of length k that occur in at least one test case $c \in C$. Let $\mathbf{e} := (e_{\mathbf{e}_1}, e_{\mathbf{e}_2}, \ldots, e_{\mathbf{e}_k}) \in \mathcal{U}_C^k$ be an arbitrary subsequence of length k. Then, we can define the *coverage of k^{th} order* of $\mathbf{e}$ in a reduced test suite S as

$$\kappa^k(\mathbf{e}, S) := |\{c \in S \,:\, \mathbf{e} \subset c\}|,$$

where in this context "$\subset$" has the meaning "is subsequence of". Let K be the length of the longest test case in $\mathcal{U}_C^k$ and $k \in \{1, \ldots, K\}$. Consequently, the corresponding mean coverage of k^{th} order is then given by

$$\overline{\kappa^k(\mathcal{U}_C^k, S)} := \frac{1}{|\mathcal{U}_C^k|} \cdot \sum_{\mathbf{e} \in \mathcal{U}_C^k} \kappa^k(\mathbf{e}, S).$$

Hence, we obtain the coverage variance of k^{th} order as

$$\mathrm{Var}^k(\mathcal{U}_C^k, S) := \sum_{\mathbf{e} \in \mathcal{U}_C^k} \frac{(\kappa(\mathbf{e}, S) - \overline{\kappa(\mathcal{U}_C^k, S)})^2}{|\mathcal{U}_C^k|}.$$

5.1.2.6 Optimization problems of higher order

The objective function can now be reformulated as

$$F_C^K(S) := \sum_{c \in S} \lambda(c) + \sum_{k=1}^{K} \mathrm{Var}^k(\mathcal{U}_C^k, S), \tag{5.4}$$

with side conditions (5.3).
Calculation of variances for $k > 1$ is computationally very complex, since the set $\mathcal{U}_C^k$ usually is unknown beforehand and has to be computed at first. For practical reasons we will therefore only consider variances up to order 2. Hence, for the remainder we will use the following objective function:

$$F_C^2(S) := \sum_{c \in S} \lambda(c) + \sum_{k=1}^{2} \mathrm{Var}^k(\mathcal{U}_C^k, S). \tag{5.5}$$

By assigning weights to the three addends above, each objective can be prioritized as desired.

5.2 Implementation of GWMSA

5.2.1 Greedy

The greedy algorithm we will use as initialization strategy for the implementation of GWMSA for the TESTSUITE problem is based on the inclusion probability distribution that will be derived for the implementation of the modified simulated annealing part in subsection 5.2.2. It starts with an empty test suite S_0 and iteratively adds test cases c from $C \setminus S_0$ to S_0 until every function $e \in \mathcal{U}$ is covered by at least one test case $c \in S_0$.
In each iteration, all test cases $C \setminus S_0$ are ordered by their inclusion probability $\mathbb{P}_{inc}(c)$ (see subsection 5.2.2), and the test case with the highest value of $\mathbb{P}_{inc}$ is added to S_0:

Input:

- Complete test suite C, set of functions $\mathcal{U}$.

Algorithm:

$\mathcal{U}_0 := \mathcal{U}$, $S_0 = \emptyset$.
while $\mathcal{U}_0 \neq \emptyset$ **do**
  $p_{max} := 0$.
  for all $c \in C \setminus S_0$ **do**
    Compute $\mathbb{P}_{inc}(c)$.
    if $\mathbb{P}_{inc}(c) > p_{max}$ **then**
      $p_{max} := \mathbb{P}_{inc}(c)$.
      $c_{next} := c$.
    end if
  end for
  $S_0 := S_0 \cup c_{next}$.
  for all $e \in c_{next}$ **do**
    $\mathcal{U}_0 := \mathcal{U}_0 \setminus e$.
  end for
end while

Output:

- S_0.

5.2.2 Modified simulated annealing

5.2.2.1 Neighborhood definition

The neighborhood definition of the GWMSA implementation for the classic set covering problem from section 3.2.3 can be adopted unchanged for the extended test suite reduction problem. Thus, we have

$$N(S) := \Bigg\{ S' \Big| \bigcup_{c' \in S'} \bigcup_{e \in c'} e = \mathcal{U}, \exists c \in S \setminus S', c_1, \ldots, c_n \in S' \setminus S \; : \\ S' = (S \setminus c) \cup c_1 \cup \ldots \cup c_n \Big\}.$$

5.2.2.2 Definition of candidate generation probabilities

Regarding our neighborhood definition from last subsection, we have to define probabilities for the inclusion and exclusion of test cases into and from a reduced test suite:
Analogous to the construction of candidate generation probabilities for the set covering problem (see section 3.2.4), it is clear that when considering the original test suite reduction problem, adding test cases c to a partial solution S_0 with a high number of yet uncovered functions $e \in \mathcal{U}$ that would be newly covered when adding c to S_0 will lead to better solutions of F_C^2 from equation (5.5). Thus, the probability for adding a test case $c \in C \setminus S_0$ to a partial solution S_0 should be proportional to

$$d(S_0, c) := |\{e \in \mathcal{U} \; : \; e \in c \wedge \kappa(e, S_0 \setminus c) = 0 \wedge \kappa(e, S_0 \cup c) = 1\}|.$$

Conversely, the probability for removing a test case $c \in S_0$ from the intermediate solution S_0 should be inversely proportional to $d(S_0, c)$.
Hence, we have

$$\mathbb{P}_{inc}(S_0, c) \propto d(S_0, c) \quad \text{and} \quad \mathbb{P}_{ex}(S_0, c) \propto \frac{1}{d(S_0, c)}. \tag{5.6}$$

As the test case length $\lambda(c)$ has a direct effect on the value of the objective function F_C^2, it is obvious that short test cases should be preferred over long test cases. Or, in other words, the probability for including a test case c to a partial solution S_0

should be inversely proportional to its length and, the other way round, the exclusion probability should be proportional to its length:

$$\mathbb{P}_{inc}(S_0, c) \propto \frac{1}{\lambda(c)} \quad \text{and} \quad \mathbb{P}_{ex}(S_0, c) \propto \lambda(c). \tag{5.7}$$

Similarly, if coincidently test cases are selected and added to S_0 in a way that individual first and second order variance contribution is minimized, this will probably lead to good solutions of F_C^2. When successively adding test cases $c \in C \setminus S_0$ to an initially empty test suite S_0, individual variance contribution can be taken into consideration by comparing $\mathrm{Var}^p(\mathcal{U}_C^p, S_0 \cup c)$ with $\mathrm{Var}^p(\mathcal{U}_C^p, S_0 \cup \tilde{c})$ for all $c \in C \setminus S_0$, $\tilde{c} \neq c$ and $p = 1, 2$. Conversely, for the removal of test cases $c \in S_0$, less values of $\mathrm{Var}^p(\mathcal{U}_C^p, S_0 \setminus c)$ will more likely decrease F_C^2.
So, the probability for removing a test case $c \in S_0$ from the intermediate solution S_0 should be inversely proportional to the variance value $\mathrm{Var}^p(\mathcal{U}_C^p, S_0 \setminus c)$ that would result when removing c from S_0. Analogously, the probability for adding a test case c to S_0 should be inversely proportional to $\mathrm{Var}^p(\mathcal{U}_C^p, S_0 \cup c)$. In summary, this means

$$\mathbb{P}_{inc}(S_0, c) \propto \frac{1}{\mathrm{Var}^p(\mathcal{U}_C^p, S_0 \cup c)} \tag{5.8}$$

and

$$\mathbb{P}_{ex}(S_0, c) \propto \frac{1}{\mathrm{Var}^p(\mathcal{U}_C^p, S_0 \setminus c)} \tag{5.9}$$

for $p = 1, 2$. So finally, when putting (5.6), (5.7), (5.8), and (5.9) together we obtain

$$\mathbb{P}_{inc}(S_0, c) \propto \frac{d(S_0, c)}{\lambda(c) + \mathrm{Var}^1(\mathcal{U}_C^p, S_0 \cup c) + \mathrm{Var}^2(\mathcal{U}_C^p, S_0 \cup c)} \tag{5.10}$$

and

$$\mathbb{P}_{ex}(S_0, c) \propto \frac{\lambda(c)}{d(S_0, c) + \mathrm{Var}^1(\mathcal{U}_C^p, S_0 \setminus c) + \mathrm{Var}^2(\mathcal{U}_C^p, S_0 \setminus c)}. \tag{5.11}$$

To construct the inclusion probability distribution from relation (5.10), we first define

$$\alpha(S_0, c) := \frac{d(S_0, c)}{\lambda(c) + \mathrm{Var}^1(\mathcal{U}_C^p, S_0 \cup c) + \mathrm{Var}^2(\mathcal{U}_C^p, S_0 \cup c)}. \tag{5.12}$$

The denominator in (5.12) can never become zero for the same reasons as in section 3.2.4 (the variances of first and second order are always nonnegative). So, for all $c \in C \setminus S_0$ we can set

$$\mathbb{P}_{inc}(S_0, c) := \frac{\alpha(S_0, c)}{\sum_{\tilde{c} \in C \setminus S_0} \alpha(S_0, \tilde{c})}. \tag{5.13}$$

For the definition of exclusion probabilties, we have to make a case differentiation again (see section 3.2.4 for details). Therefore, we first redefine the set Γ_0 by

$$\Gamma_0 := \{c \in S_0 \,:\, d(S_0, c) + \mathrm{Var}^1(\mathcal{U}_C^p, S_0 \setminus c) + \mathrm{Var}^2(\mathcal{U}_C^p, S_0 \setminus c) = 0\}.$$

For the special case that $\Gamma_0 \neq \emptyset$ we define exclusion probabilities as follows:

$$\begin{aligned} \mathbb{P}_{ex}(S_0, c) &:= \frac{\lambda(c)}{\sum_{\tilde{c} \in \Gamma_0} \lambda(\tilde{c})}, \quad \text{if} \quad c \in \Gamma_0 \quad \text{and} \\ \mathbb{P}_{ex}(S_0, c) &:= 0, \quad \text{else.} \end{aligned} \tag{5.14}$$

For the regular case that $\Gamma_0 = \emptyset$ we can define the exclusion probability distribution analogously to the inclusion probabilities by

$$\mathbb{P}_{ex}(S_0, c) := \frac{\beta(S_0, c)}{\sum_{\tilde{c} \in S_0} \beta(S_0, \tilde{c})}, \tag{5.15}$$

where

$$\beta(S_0, c) := \frac{\lambda(c)}{d(S_0, c) + \mathrm{Var}^1(\mathcal{U}_C^p, S_0 \setminus c) + \mathrm{Var}^2(\mathcal{U}_C^p, S_0 \setminus c)}.$$

Altogether, we obtain the following definition of candidate generation probabilities:

$$\mathbb{P}_{cg}(S, S') := \begin{cases} \frac{\lambda(c)}{\sum_{\tilde{c} \in \Gamma_0} \lambda(\tilde{c})}, & \text{if} \quad c \in \Gamma_0 \wedge S' = S \setminus c \\ 0, & \text{else} \end{cases},$$

if $\Gamma_0 \neq \emptyset$ and

$$\mathbb{P}_{cg}(S, S') := \begin{cases} \mathbb{P}_{ex}(S, c) \cdot \prod_{i=1}^{n} \mathbb{P}_{inc}((S \setminus c) \bigcup_{j=1}^{i-1} c_j, c_i) & \text{if} \quad S' = (S \setminus c) \bigcup_{i=1}^{n} c_i \\ 0, & \text{else} \end{cases},$$

if $\Gamma_0 = \emptyset$.

Hence, in the same way as for the set covering problem (revise chapter 3.2 for details), for implementation of the modified simulated algorithm for TESTSUITE, a case differentiation whether Γ_0 is empty or not has to be integrated additionally into the abstract specification from subsection 3.1.2.3, before a neighbor solution candidate is drawn.

5.3 Empirical simulation

In this section we will compare the performance of our GWMSA implementation from section 5.2.2 in combination with the greedy algorithm in section 5.2.1 for the TESTSUITE problem with two other greedy approaches and a brute force strategy.

The performance evaluation is done on 13 different representative problem instances. We will see that our GWMSA algorithm is able to determine the optimal solution for most of the representative problem instances within a small amount of additional computational effort compared to the other greedy based approximation strategies. These greedy strategies, however, mostly only yield to suboptimal solutions.

5.3.1 Data setup

As input for our experimental studies of the TESTSUITE problem we derived 13 different transition state machines which were designed on the basis of industrial case studies. To get realistic statements for the context of our studies, most of our use cases are small- or intermediate-sized (I-IX). Nevertheless, we included some larger models as well (X-XIII).

	$\|C\|$	$H(C)$	$H(C^2)$	$\|\mathcal{U}\|$	$\|\mathcal{U}_C^2\|$
I	15	84	69	13	26
II	21	123	102	10	15
III	32	128	96	13	28
IV	41	164	123	15	31
V	30	189	159	25	35
VI	36	190	154	31	40
VII	46	317	271	40	52
VIII	45	374	329	23	36
IX	120	600	480	15	33
X	132	1306	1174	26	40
XI	512	6656	6144	30	54
XII	284	1600	1316	140	422
XIII	625	4375	3750	23	86

Table 5.1 Use cases

For detailed information about the use cases consider table 5.1. It contains the number of test cases ($|C|$), the number of function calls ($H(C)$), the number of function-pair calls

$$H(C^2) := \sum_{c \in C} (\lambda(c) - 1),$$

the number of functions ($|\mathcal{U}|$) and the number of function-pairs ($|\mathcal{U}_C^2|$) for each of our use cases. Nevertheless, the full model definitions must not be published due to legal reasons.

5.3.2 Benchmark strategies

5.3.2.1 Successive greedy strategy

We will compare the performance of the GWMSA algorithm with another greedy strategy that successively optimizes with respect to the number of function calls first and with respect to the variance afterwards. Similar to the basic greedy algorithm for the set covering problem from [36], in every step the successive greedy algorithm computes the set of all test cases $c := \{e_{c_1}, e_{c_2}, \ldots, e_{c_{|c|}}\}$, for which the ratio of the number of additionally covered functions $d(c)$ and the test case length $\lambda(c)$ is maximal. Then it picks the one where

$$\mu_{S_0}^{(k)}(c) := \sum_{i=k}^{|c|} \kappa^k((e_{c_{i-k+1}}, \ldots, e_{c_i}), S_0)$$

is minimal for $k = 1$ and $k = 2$. If there exists more than one test case with this property, then one of them is chosen randomly. The chosen test case is afterwards added to the reduced test suite S_0. The algorithm stops as soon as all functions are covered by at least one test case in S_0.

5.3.2.2 Multiobjective greedy strategy

The second benchmark algorithm is another greedy strategy that is almost identical with the one that is used for the initialization step of GWMSA. More precisely, the greedy algorithm from section 5.2.1 and the multiobjective greedy strategy only differ in the way how the inclusion probability is defined. For the multiobjective greedy algorithm we have

$$\mathbb{P}_{inc}(c) := 0.5 \cdot \mathbb{P}_{rate}(c) + 0.5 \cdot \mathbb{P}_{var}(c),$$

where

$$\mathbb{P}_{rate}(c) := \frac{d(c)/\lambda(c)}{\sum_{\tilde{c} \in C \setminus S_0} d(\tilde{c})/\lambda(\tilde{c})}$$

and

$$\mathbb{P}_{var}(c) := \frac{1 - \frac{\mathrm{Var}^1(\mathcal{U}, S_0 \cup c)}{\sum_{\tilde{c} \in C \setminus S_0} \mathrm{Var}^1(\mathcal{U}, S_0 \cup \tilde{c})}}{|\{\tilde{c} \in C \setminus S_0\}| - 1}$$

for all $c \in C \setminus S_0$.

5.3.2.3 Brute force

The third benchmark strategy is the brute force search for the reduced test suite $S_0 \subset C$ that minimizes F_C^2 from (5.5), such that all functions are called at least once.

5.3.3 Simulation results

For our simulation studies we have applied the GWMSA implementation for TEST-SUITE on the problem instances from table 5.1. We will compare the results with those of the two greedy algorithms (successive and multiobjective) introduced above as well as with the brute force strategy.
All computations were performed on an AMD Opteron (tm) Quad Core with 2.6 GHz and 32 Gigabytes of RAM.
To determine the best combination of input parameters for modified simulated annealing, we have generated 1000 input parameter combinations and applied each of them on 100 randomly generated training instances. In doing so, the following combination has shown the best performance and was therefore used for all further computations (the termination threshold ε was again fixed at 0.001):

- Initial temperature $T_{init} = 26.23$.
- Number of runs with constant temperature $\xi = 66$.
- Temperature degression coefficient $\delta = 0.01$.

The objective function values F_C^2 that result when plugging in the individual solutions computed by the different algorithms into formula (5.5) are displayed in table 5.2.

C	MOG	SucG	BF	GWMSA	G/M	G/L	G/B
I	16.66	16.66	16.66	16.66	1	1	1
II	27.67	27.38	18.33	18.33	0.66	0.67	1
III	38.15	37.57	37.57	37.57	0.98	1	1
IV	36.21	36.07	35.94	36.01	0.99	1	1
V	43.25	43.25	39.84	39.84	0.92	0.92	1
VI	48.94	48.94	45.36	45.36	0.93	0.93	1
VII	70.78	70.78	65.21	65.21	0.92	0.92	1
VIII	62.70	62.70	62.66	62.70	1	1	1
IX	26.40	26.40	26.40	26.40	1	1	1
X	45.55	45.52	44.48	44.48	0.98	0.98	1
XI	67.15	67.11	53.37	53.41	0.8	0.8	1
XII	518.90	518.93	498.73	504.56	0.97	0.97	1.01
XIII	37.15	37.15	37.15	37.15	1	1	1

Table 5.2 Objective function values

Legend for table 5.2:

- C: Problem instances (taken from table 5.1).
- MOG: Multi objective greedy algorithm.
- SucG: Successive greedy algorithm.
- BF: Brute force.
- GWMSA: Modified simulated annealing with greedy initialization.
- G/M, G/S, G/B: GWMSA result divided through the results of MOG, SucG, and BF.

The corresponding computation times for every approach on each problem instance are displayed in table 5.3.

C	MOG	SucG	BF	GWMSA
I	< 1s	< 1s	< 1s	< 1s
II	< 1s	< 1s	4s	< 1s
III	< 1s	< 1s	4h	< 1s
IV	< 1s	< 1s	8h	< 1s
V	< 1s	< 1s	2h	< 1s
VI	< 1s	< 1s	17h	1s
VII	< 1s	< 1s	1h	< 1s
VIII	< 1s	< 1s	2h	< 1s
IX	< 1s	< 1s	1h	< 1s
X	< 1s	< 1s	9h	< 1s
XI	< 1s	< 1s	> 1d	1s
XII	< 1s	< 1s	> 1d	26s
XIII	< 1s	< 1s	12h	1s

Table 5.3 Computation times

Legend for table 5.3:

- s: seconds.
- h: hours.
- d: days.

We can see that for all except for one problem instance, GWMSA is able to approximate the global optimum computed by BF exactly or nearly exact in one second at most (see last columns in tables 5.2 and 5.3).
When comparing the results of GWMSA with those of the greedy approximations MOG and SucG (see sixth and seventh column in table 5.2) we can see that GWMSA is able to improve the results of MOG and SucG by 7% and 6% on average, respectively, within a minimal additonal effort of computational time, except for the penultimate problem instance XII, where the additional effort (26 seconds) and the result improvement (3%) show an inappropriate interrelation due to the complexity of this problem instance. For the problem instances II and XI, greedy results can even be improved by over 20%.

In summary, we can conclude that application of GWMSA on the problem of test suite reduction with balancing constraint is reasonable as the results of greedy approximations like MOG and SucG can be improved with respect to the objective

function in the majority of cases within a minimal additional computational effort in almost all cases. One interesting question for future research is the determination of the approximation bound for the greedy algorithm designed for computation of an initial solution for modified simulated annealing. Besides, the results should be validated by applying the algorithms on a greater number of representative problem instances. Finally, it should be investigated how another practical requirement, namely a timeout constraint (maximum computation time limit for determination of a reduced test suite) can be taken into account.

6 Task allocation

6.1 Problem specification

6.1.1 Introduction

Due to global climate warming, the number of natural disasters has increased critically in the last years. Present examples are the great forest fires in New Mexico (USA) in June 2011 or the series of tornados in the USA in May 2011 which have badly affected many cities across the state of Missouri. Especially in cases of great fires or floodings, the fire brigades carry enormous responsibility. Hence, when facing a conflagration (or a flooding), a diligent distribution of tasks between fire fighters is crucial for an efficient preservation from major damage.
Computing such a task distribution is a non-trivial challenge in general, since one task usually can be coped by many different fire brigade roles (also known as "functions") like water squad or attacking squad.
However, the physical work load when performing a specific task is different for the given roles. For example, the attacking squad can prepare the water supply very quickly in two minutes but is much more stressed than the water squad when performing this task. The water squad, in turn, usually needs four minutes until extinguishing water will be finally available. So, on one side it is clear that task distribution has to be efficient in a way that duration of the whole fire fighting procedure is minimized. On the other hand, the total work load of every role must be balanced, since excessive stress of one specific role may lead to (unintended) misconduct with severe impacts especially in life-threatening situations.
Nowadays in general, the on-scene commander allocates the necessary tasks like establishing water supply or bringing the water pump into service to fire brigade roles based on his experience. This practice is efficient and reasonable for ordinary events like house fires. Furthermore, a balanced workload can be guaranteed in most cases. However, the heads of operations usually only have minor or even no experience with major damage situations like the ones described above. On the other hand, when fighting for example a conflagration, fire fighters often work many hours at a stretch without any break. Especially in these situations, a balanced workload of fire brigade roles with respect to their physical and psychic stress level they are exposed to is crucial but cannot be ensured due to missing experience.
In this chapter we will formalize the problem of task assignment to fire brigade roles as a binary integer program. Furthermore, similar to the CASASSIGN problem, we suggest the quantification of work load by functions that assign a stress level which will serve as a penalty value to each fire brigade role depending on the working time.

Given a set of available fire brigade roles and a list of tasks that have to be carried out, the formal goal is then to minimize the global penalty over all fire brigade roles. Later, we will also show that this binary integer program is again another generalization of the set covering problem and thus $\mathcal{NP}$-complete.

6.1.2 Formal representation of task allocation

In this subsection we will formally define the task allocation problem as a binary integer optimization problem and show its relationship to the set covering problem.

6.1.2.1 Optimization problem

For an instance of the task allocation problem, the following input parameters have to be given:

- Set of fire brigade roles

$$\mathcal{R} := \{r_1, \ldots, r_m\}.$$

- Set of tasks

$$\mathcal{T} := \{\tau_1, \ldots, \tau_n\}.$$

- Qualification function

$$\rho : \mathcal{R} \longrightarrow \mathcal{P}(\mathcal{T})$$

  that maps every role $r \in \mathcal{R}$ to the set of tasks, r is qualified to accomplish.

- List of processing times for a given role and a given task

$$D : \mathcal{R} \times \rho(\mathcal{R}) \longrightarrow \mathbb{R}_+.$$

- Set of strictly monotonically increasing stress (penalty) functions

$$\Pi := \{\pi_1, \ldots, \pi_m\},$$

  where $\pi_i : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$ for all $i = 1, 2, \ldots, m$, and π_i maps total working time t_i of role r_i to a penalty value.

Next, we define decision variables $\mathcal{X} := (x_{ij})_{i \in \mathcal{I}, j \in \mathcal{J}}$ with $\mathcal{I} = \{1, 2, \ldots, m\}$ and $\mathcal{J} = \{1, 2, \ldots, n\}$ with meaning

$$x_{ij} = 1 \Leftrightarrow \text{ task } \tau_j \text{ is performed by role } r_i,$$
$$x_{ij} = 0 \Leftrightarrow \text{ task } \tau_j \text{ is not performed by role } r_i.$$

Then, the optimization problem can be defined as follows:

- Minimize

$$F(\mathcal{X}) := \sum_{i=1}^{m} \pi_i \left(\sum_{j\,:\,\tau_j \in \rho(r_i)} D(r_i, \tau_j) \cdot x_{ij} \right), \tag{6.1}$$

- such that

$$\sum_{i\,:\,\tau_j \in \rho(r_i)} x_{ij} \geq 1, \tag{6.2}$$

  for every $j = 1, 2, \ldots, n$.

Remarks:

- In formula (6.1) the total penalty (corresponds to physical work load) of all roles is summarized. This sum should be minimized, since we want to minimize the physical work load of all roles.
- By fulfilling inequalities (6.2) it is ensured that each task $\tau_1, \ldots, \tau_n \in \mathcal{T}$ is performed by at least one role $r \in \mathcal{R}$.
- It is clear that tasks cannot be assigned to a role that is not qualified to accomplish them. Formally, this means if $\tau_j \notin \rho(r_i)$, then necessarily $x_{ij} = 0$.
- Given a configuration $\mathcal{X}$ that satisfies (6.2), for $i = 1, 2, \ldots, m$, the total working time t_i of role r_i can be derived by

$$t_i := t_i(\mathcal{X}) = \sum_{j\,:\,\tau_j \in \rho(r_i)} D(r_i, \tau_j) \cdot x_{ij}.$$

  Thus, the value of the global objective function F can alternatively be expressed subject to the working time of every role as follows:

$$F(\mathcal{X}) = F(t_1, t_2, \ldots, t_m) = \sum_{i=1}^{m} \pi_i(t_i).$$

In the following subsections we call $S := \{(r, \tau)_{r \in \mathcal{R}, \tau \in \mathcal{T}}\}$ an *assignment* of tasks to roles. S is said to be *feasible* if every task $\tau \in \mathcal{T}$ is contained in exactly one tuple in S and if for every $(r, \tau) \in S$ we have $r \in \rho^{-1}(\tau)$.
We assume that every task can be accomplished by a qualified role on its own without any help by another role. This assumption is unproblematic for our application on fire fighting, since in general, tasks in the context of fire fighting procedures

are defined in a way, such that they always can be coped by one single role. An assignment S where every task $\tau \in \mathcal{T}$ is assigned to at least one qualified role would also be feasible, of course. However, it is clear that an assignment that contains a task $\tau \in \mathcal{T}$ which is assigned to more than one qualified role can never be optimal with respect to (6.1), as divesting τ from all but one qualified role will always lead to an assignment that is still feasible and corresponds to a smaller value of F. In the remainder, we will therefore only consider assignments where every task is assigned to exactly one qualified role.
If we set

$$x_{ij} = 1 \Leftrightarrow (r_i, \tau_j) \in S \quad \text{and} \quad x_{ij} = 0 \Leftrightarrow (r_i, \tau_j) \notin S,$$

then every (not necessarily feasible) assignment S corresponds to an assignment of the decision variables in $\mathcal{X}$. We will denote this related assignment by $\mathcal{X}_S$.

Using the notation in section 2.1, the set of components C contains all tuples (r, τ) of tasks $\tau \in \mathcal{T}$ and roles $r \in \mathcal{R}$ that are qualified to perform task τ. The solution space $\mathcal{S}$ of TASKALLOC is given by those collections $S \subset C$ where every task is assigned to one qualified role.

6.1.2.2 Relationship between task allocation and set cover

Given an arbitrary set covering problem instance composed of equations (3.4) and (3.5), we can identify every set $c_i \in C$ with a role $r_i \in \mathcal{R}$ and define the qualification map ρ by $\rho(r_i) := c_i$. Next, we define the set of decision variables $\mathcal{X}$ as

$$x_{ij} := x_i \quad \text{for all } i = 1, 2, \ldots, m \text{ and } j = 1, 2, \ldots, n.$$

Then, we identify the processing times with the cost function,

$$D(r_i, \tau_j) := \lambda(c_i) \quad \text{for all } i = 1, 2, \ldots, m \text{ and } j = 1, 2, \ldots, n,$$

and set all penalty functions equal to identity, that is

$$\pi_i(t) := t \quad \text{for all } i = 1, 2, \ldots, m \text{ and } t \in \mathbb{R}_{\geq 0}.$$

So finally, we have created a special instance of our task allocation problem (6.1) with side conditions (6.2), and thus, this problem can be seen as a generalization of set cover.

6.2 Implementation of GWMSA

6.2.1 Greedy

The greedy strategy for the TASKALLOC problem is also very straightforward. Given a set of roles $\mathcal{R}$ and a set of tasks $\mathcal{T}$, the greedy algorithm iteratively assigns an open task to one qualified role, such that increase of global penalty in every step is minimized:

Input:

- List of roles $\mathcal{R} = \{r_1, r_2, \ldots, r_m\}$.
- List of tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$.
- Qualification function ρ.
- List of processing times D.
- Role specific penalty functions $\pi_1, \pi_2, \ldots, \pi_m$.

Algorithm:

$F_0 := 0$.
$S_0 := \emptyset$.
$\mathcal{T}_0 := \{1, 2, \ldots, n\}$.
while $\mathcal{T}_0 \neq \emptyset$ **do**
  $\Delta F_{min} := \infty$.
  for all $j \in \mathcal{T}_0$ **do**
    for all $i \in \{k \mid \tau_j \in \rho(r_k)\}$ **do**
      $S_0 := S_0 \cup \{(r_i, \tau_j)\}$.
      $F_{new} := F(\mathcal{X}_{S_0})$.
      $S_0 := S_0 \setminus \{(r_i, \tau_j)\}$.
      if $F_{new} - F_0 < \Delta F_{min}$ **then**
        $I := i$.
        $J := j$.
        $\Delta F_{min} := F_{new} - F_0$.
      end if
    end for
  end for
  $F_0 := F_0 + \Delta F_{min}$.
  $S_0 := S_0 \cup \{(r_I, \tau_J)\}$.
  $\mathcal{T}_0 := \mathcal{T}_0 \setminus J$.
end while
$S := S_0$.

Output:

- Feasible assignment S.

6.2.2 Modified simulated annealing

6.2.2.1 Neighborhood definition

We call two feasible assignments S_1 and S_2 *neighbored* if they differ from each other in the assignment of exactly one task $\tau \in \mathcal{T}$. This means that $S_1 \setminus S_2 = (r_i, \tau)$ and $S_2 \setminus S_1 = (r_j, \tau)$ for different roles $r_i, r_j \in \rho^{-1}(\tau)$ with $i, j \in \{1, 2, \ldots, m\}$ and $i \neq j$. Generally,

$$N(S) := \{S' \in \mathcal{S} \,|\, \exists \tau \in \mathcal{T}, r, r' \in \rho^{-1}(\tau) \,:\, S \setminus S' = (r, \tau) \wedge S' \setminus S = (r', \tau)\}.$$

6.2.2.2 Definition of candidate generation probabilities

In each simulated annealing iteration, one task is randomly chosen and reassigned to a different role to create a neighbored assignment. Hence, in one iteration the following two random operations have to be performed:

1. Randomly draw a task $\tau \in \mathcal{T}$ that will be reassigned.

2. Randomly draw a qualified role $r \in \rho^{-1}(\tau)$ where τ will be assigned to.

It is clear that reassigning tasks that cause a large share of current global penalty will probably have a greater effect than reassigning lightweight tasks. We will therefore draw a task with a probability that is proportional to this share.
For a given feasible assignment S and arbitrary $J \in \{1, 2, \ldots, n\}$ let $\varrho(\tau_J) \in \mathcal{R}$ be the role that is currently assigned to task $\tau_J \in \mathcal{T}$ in S, so $(\varrho(\tau_J), \tau_J) \in S$. Let $S_J := S \setminus (\varrho(\tau_J), \tau_J)$. Then, the difference $(F(\mathcal{X}_S) - F(\mathcal{X}_{S_J}))$ corresponds to the global penalty share of task τ_J. Furthermore, we have

$$\sum_{j=1}^{n} (F(\mathcal{X}_S) - F(\mathcal{X}_{S_j})) = F(\mathcal{X}_S),$$

as the sum over all global penalty shares of all tasks must be equal to the global penalty itself again. Hence, for every task τ_j, $j = 1, 2, \ldots, n$ we can define the reassignment probability as follows:

$$\mathbb{P}_1(\tau_j) := \frac{F(\mathcal{X}_S) - F(\mathcal{X}_{S_j})}{F(\mathcal{X}_S)}.$$

Vice versa, assigning a task to a qualified role that minimizes increase of global penalty will more probably lead to better solutions. Let $j \in \{1, 2, \ldots, n\}$ be arbitrary and $S_j := S \setminus (\varrho(\tau_j), \tau_j)$ be an incomplete assignment where task τ_j is not assigned to any role. Furthermore, let $S_{i|j} := S_j \cup (r_i, \tau_j)$ be a feasible assignment where task τ_j is assigned to role r_i. The idea is to compare the hypothetical increases of global penalty when assigning τ_j to a specific role $r \in \rho^{-1}(\tau_j)$ that is qualified to perform τ_j. For every $i \in \{k \,|\, \tau_j \in \rho(r_k)\}$ this hypothetical increase is given by

$$\Delta(r_i, \tau_j) := F(\mathcal{X}_{S_{i|j}}) - F(\mathcal{X}_{S_j}).$$

Obviously, for every $j \in \{1, 2, \ldots, n\}$ and $i \in \{k \,|\, \tau_j \in \rho(r_k)\}$, $\Delta(r_i, \tau_j)$ is strictly positive, since the processing time for task τ_j is strictly positive for every qualified role. Thus, the global penalty when having assigned τ_j to some qualified role must be greater than the value that results when not having assigned τ_j to any role.
A corresponding probability distribution that reproduces these hypothetical penalty increases can be obtained by setting

$$\mathbb{P}_2(r_i \,|\, \tau_j) := \begin{cases} \frac{\Delta(r_i, \tau_j)}{\sum_{\{k \,|\, \tau_j \in \rho(r_k)\}} \Delta(r_k, \tau_j)}, & \text{if} \quad \tau_j \in \rho(r_i) \\ 0, & \text{else} \end{cases}.$$

Finally, the candidate generation probabilities for the modified simulated annealing algorithm are given by

$$\mathbb{P}_{cg}(S, S') := \begin{cases} \mathbb{P}_1(\tau) \cdot \mathbb{P}_2(r' \,|\, \tau), & \text{if} \quad \exists \tau \in \mathcal{T}, r, r' \in \rho^{-1}(\tau) \,:\, S' = (S \setminus (r, \tau)) \cup (r', \tau) \\ 0, & \text{else} \end{cases}.$$

6.3 Empirical simulation

In our experimental studies about the task allocation problem we will compare the performance of our GWMSA implementation from sections 6.2.1 and 6.2.2 on an empirical problem instance with action plans that were recommended by experts from the fire brigade. Before discussing the results, we will first introduce our empirical data set in detail.

6.3.1 Data setup

We will first identify the task and role sets in the following subsection and derive role specific penalty functions afterwards.

6.3.1.1 Tasks and roles

For our experiments we have collected lists of tasks that were considered as relevant within fire fighting of conflagrations by 6 different operations managers from fire brigades in Germany and asked them which roles are able to cope with which tasks. We also asked them for corresponding processing time estimations for each task and qualified role. In doing so, we were able to identify 8 different main tasks that are split up into 37 subtasks in total. The main tasks in the context of fire fighting are

- approaching to the source of fire (for example proceeding in the stairways of buildings),
- equipping the roles (especially attacking and safety squad),
- bringing the water pump into service,
- establishing and monitoring respiratory protection,
- establishing water supply (from fireplug to vehicles and from vehicles to manifold),
- placing water manifold,
- reconnaissance,
- issue of orders of managing roles to carrying out roles.

In general, the available action forces to accomplish the tasks involve the following 12 roles:

- B service (BS),
- C service (CS),
- station officer (SO),
- first squadron leader (SL1),
- second squadron leader (SL2),
- first attacking squad (AS1),
- second attacking squad (AS2),
- first water squad (WS1),

- second water squad (WS2),
- safety squad (SS),
- first engineer (E1),
- second engineer (E2).

Furthermore, there are 4 additional action forces available (MF1, MF2, MF3, and MF4) that can take miscellaneous roles. Note that not necessarily every available action force has to be commited. Sometimes this is even not reasonable if, for example, workflows cannot be parallelized with the consequence that one or more involved forces would be idle.
Usually, the tasks with organizing character like reconnaissance and issuing of orders can be performed by the different groups of managing roles, where for example station officers and squadron leaders belong to. On the other hand, the executing roles like attacking and water squad are at command for performing physical tasks like establishing water supply. Also at this, the distribution of physical tasks to the different qualified roles is ambiguous.
The 37 subtasks and 16 roles from above will define the sets $\mathcal{T}$ and $\mathcal{R}$ for our experiments that will be described later in subsection 6.3.3. Furthermore, by using the time estimations given by the experts we were able to define the qualification function ρ and processing times D. The derivation of corresponding penalty functions π_1 to π_{16} for the different roles will be described in detail in the next subsection.

6.3.1.2 Derivation of penalty functions

In this subsection we will derive role specific penalty functions which assign a penalty value to a role depending on its working time. For this purpose, we will first define categories of tasks and use the increase of body temperature when performing the tasks of one category as measure of stress. Out of this, we can derive category specific penalty functions, and finally, we classify every role into a category. So, in the end, every role is assigned a penalty function that corresponds to its area of task responsibility, and for roles whose spectrum of tasks is exceedingly stressful working time is penalized to a greater extent than for other roles.

a. Introduction

During fire fighting, fire fighters are exposed to physiological and psychical stress factors: The lack of time caused by threatened occupants [28], working under perilous conditions [120], the transport of additional weights and limited mobility due to personal protective equipment [59]. Those stress factors affect the development

time in fire fighting operations. Therefore, it is necessary to consider these influences when deriving penalty functions for the different involved roles.

b. Tasks and their surrounding conditions

It has been shown that anticipated tasks during firefighting operations vary in consideration of work intensity, dangerousness, and expert knowledge. Hence, it is necessary to specify the expected tasks in terms of their impact on the work capability of fire fighters. Based on German standing order procedures for compartment fire fighting [8], [7], defined tasks have been differentiated by means of the operator, the protective equipment as well as physiological and psychic stress factors:

- **Directing the fire fighting**.
  - *Operator:* Crew captain.
  - *Equipment:* Protective clothing and helmet.
  - *Physiological stress factors:* Staying off the building, no transport of additional weights.
  - *Psychic stress factors:* Actions on scene and being self dependent.
- **Fire fighting or/and evacuating victims**.
  - *Operator:* Fire fighting crews.
  - *Equipment:* Protective clothing, helmet, and self-contained breathing apparatus (SCBA).
  - *Physiological stress:* Entering the building, transport of additional weights.
  - *Psychical stress:* Actions on scene, exposuring oneself to dangers like heat, and limited breathing air.
- **Operating water hoses, driving, and operating the pumper truck or SCBA control devices and supporting the fire fighting crews**.
  - *Operator:* Exterior fire fighters.
  - *Equipment:* Protective clothing and helmet.
  - *Physiological stress:* Staying off the building, transport of additional weights.
  - *Psychic stress:* Actions on scene.

Deduced from this differentiation, penalty functions must be found, subject to the regarded operator and his determined impact factors. The next step is to identify parameters permitting the quantification of qualitative formulated impacts on the operator's work capabilities mentioned above.

c. Impact of work intensity

Former studies [128], [66] have shown that fire fighting affects the cardiovascular function as well as the required strength for prolonged duration. The occupation during a shift is characterized by inactivity and a sudden change to high physical stress expressed by heart rate (HR) [75]. It has been determined that an immediate respond to an alert tends to already increase heart rates [15]. After the arrival on the scene of a fire, prolonged periods of low- intensity efforts alternate with moderate- and even high-intensity efforts [25]. Barnard and Duncan [15] have identified that physical strain reaches close to maximum values. To complete suppression duties, levels of about 80% of the maximal heart rate (HR_{max}) have been reported by Sothman [128]. But due to the influence of heat stress, the heart rate is not a sufficient and accurate measure to completely describe energy expenditure [119].

Maximum oxygen consumption ($VO_{2\ max}$) has been recognized as another important factor to evaluate the fire fighter's work intensity. Performing simulated fire fighting duties, 60% to 80% $VO_{2\ max}$ have been measured by Lemon and Hermiston [101]. The weight and the insulating properties of protective clothing implicated increasing oxygen requirements and further increasing heart rates as well as a high environmental heat strain [53]. Elsner and Kolkhorst [56] have ascertained that individuals with lower $VO_{2\ max}$ levels take a longer time to complete defined tasks than counterparts with higher $VO_{2\ max}$ levels. Studies from Sothman [127] have proposed that an aerobic power value of $33.5ml \cdot kg^{-1} \cdot min^{-1}$ should be the minimum accepted for performing fire fighting activities.

Based on the maximum oxygen consumption, it is possible to assess different tasks in consideration of the work intensity. Furthermore, the maximum time of SCBA usage can be estimated. It would be wrong to conclude that decreasing fire fighter capability depends on increasing oxygen consumption. Due to an increasing oxygen consumption, higher work loads are rather sufferable for prolonged time periods. As recently as the individual fitness level is exhausted, decreasing developments would be expected. Assuming a progressive work intensity, a mathematical function describing the fire fighter's work capability based on oxygen consumption would probably first ascend but then consequently descend after some time. Contemporaneously, the lapse of oxygen consumption would increase fast at the beginning and approach a maximum value in the last section. Considering the unknown inflexion

point in the fire fighter's work capability curve and the unknown time to the maximum oxygen consumption, it is not useful to deduce penalty functions directly from the maximal oxygen consumption.

During every activity, heat is produced by the body according to the work intensities [5]. As mentioned by Rossi [120], surplus energy can be transferred by respiration (10%), the release of dry (70%), and evaporative heat through the skin (20%). Protective clothing properties worn by fire fighters have been described as heavy, thick, and with multiple layers. Therefore, the heat exchange under environmental conditions is limited due to the reduced water-vapour permeability. If a fire fighter equipped with protective clothing is affected by the consequences of strenous exercise in high ambient temperatures, this leads to high levels of cardiovascular and thermoregulatory strain. In addition to increasing oxygen requirements and heart rates, increasing core temperatures T_{core} have been quantified, even in case of atmospheric ambient temperatures [53]. Higher core temperatures lead to higher levels of dehydration resulting in the deterioration of cognitive performances [37]. Rossi [120] has already concluded that if the maximum allowed core temperatures are known, the maximum working times for each task can be defined. Former studies have shown that the rectal temperature is used to identify the risk of thermal injuries. Rectal temperatures higher than 39.2° Celsius lead to total disability (T_{crit}) [141]. According to Kenney [96], heat stroke occures (T_{stroke}) at 40.6° Celsius. Kenney has also reported that rectal temperatures (T_{death}) below $42° - 44°$ Celsius are sustainable to avoid death. However, based on increasing core temperatures, a decreasing fire fighter capability can be observed if core temperature limits are taken into account as mentioned above.

d. Determination of penalty function types

Data presented by Brown and Stickford [28] have been raised at real fire scenes. Heart rates and breaths per minute have been measured for tasks like fire fighting, ventilation, and search. Based on these data, the physiology of structural fire fighting has been shown. Developing penalty functions is not possible, because T_{core} values were not part of this study.
Former studies [84], [21] have taken core temperatures during the training of fire fighting activities into account but have not distinguished between different tasks. Romet [119] has measured rectal temperatures during fire fighting at a training facility and has noted that physical and environmental stresses of the various activities leaded to an increase of T_{rec}. Furthermore, the T_{rec} time curves for formulated tasks are given. Figure 6.1 shows the T_{rec} time curves of one fire fighter carrying out two different tasks.

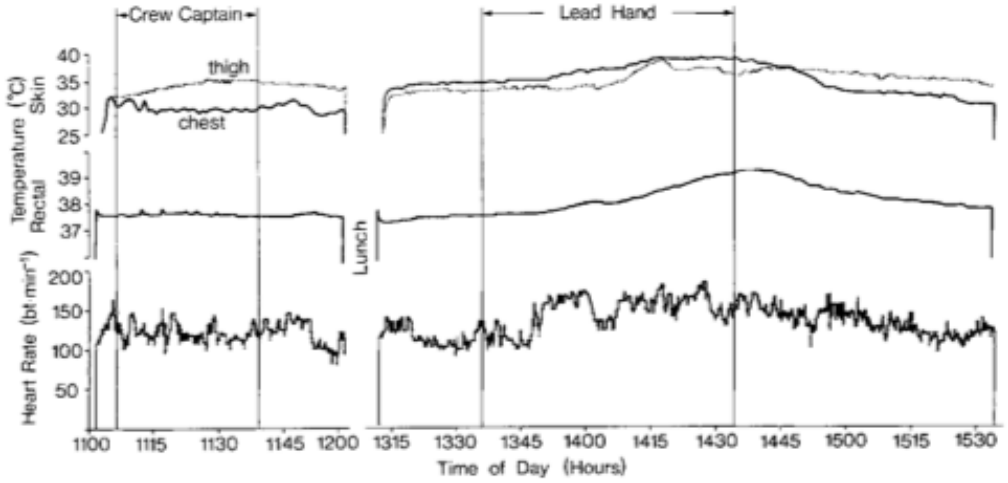


Figure 6.1 Time curve of HR and T_{rec} during fire fighting activities [119]

For six fire fighting scenarios overall 23 man-runs were collected. The procedure of this experiment has been formulated by Romet as follows: A group of five fire fighters with a pumper truck was responded to an alarm, approached the building, evaluated the situation, searched for and evacuated victims, and extinguished the fires. Here, every fire fighter has been classified into one out of four categories:

1. *Lead hand (LH)* (fire fighting or/and evacuating victims).
2. *Secondary help (SH)* (fire fighting or/and evacuating victims, at a later time also supporting the lead hand).
3. *Exterior fire fighting (EF)* (operating water hoses, driving and operating the pumper truck).
4. *Crew captain (CC)* (directing the fire fighting).

The temperatures $T_{rec\ initial}$ and $T_{rec\ end}$ before and after having performed the tasks mentioned above were measured for every category. The resulting values are displayed in table 6.1.

Group	**LH**	**SH**	**EF**	**CC**
Duration (in minutes)	24.2	19.8	23.3	28.7
$T_{rec\ initial}$ (in degrees Celsius)	37.7	37.7	37.6	37.6
$T_{rec\ end}$ (in degrees Celsius)	39.0	38.4	38.0	37.9

Table 6.1 T_{rec} during different fire fighter activities [119]

Due to the missing values between $T_{rec\ initial}$ and $T_{rec\ end}$, regression equations were developed. Based on figure 6.1, the time curve of T_{rec} has been assumed as an exponential function. The following equations could be determined:

- *Lead hand:*

$$T_{LH}(t) = 0.83098 * exp(0.039239 * t) + 36.869. \tag{6.3}$$

- *Secondary help*

$$T_{SH}(t) = 0.81336 * exp(0.031049 * t) + 36.887. \tag{6.4}$$

- *Exterior fire fighting*

$$T_{EF}(t) = 0.98191 * exp(0.014857 * t) + 36.618. \tag{6.5}$$

- *Crew captain*

$$T_{CC}(t) = 0.95856 * exp(0.0093882 * t) + 36.641. \tag{6.6}$$

As already noted in the last paragraph, no reasonable work can be accomplished anymore if T_{rec} exceeds 39.2 degrees Celsius. Refering to equations (6.3) to (6.6), the critical working duration t_{crit} until this temperature value is reached depends on the role category. When plugging in 39.2 on the left hand side of equations (6.3) to (6.6) we obtain the following critical working durations (in minutes) for the different role categories:

- *Lead hand:*

$$t_{crit,LH} = 26.2862690654.$$

- *Secondary help*

$$t_{crit,SH} = 33.660563827.$$

- *Exterior fire fighting*

$$t_{crit,EF} = 65.0750432084.$$

- *Crew captain*

$$t_{crit,CC} = 104.59296540.$$

We can see that for the lead hand and the secondary help the critical body temperature is already reached after about half an hour of physical activity, whereas the exterior fire fighting and the crew captain role categories can work for more than one

hour whithout suffering critical body heat. Hence, it is clear that roles that belong to one of the first two categories have to be spared within an allocation of tasks to roles. To achieve this, we suggest to assign penalty functions to the different role categories with respect to their resilience.

Clearly, we do not want to rule out that there might exist many other reasonable ways to incorporate this constraint, and any strategy to derive the final definition of the penalty functions appears to be some kind of haphazardly. However, in this work we just want to point out the versatility of our penalty function concept. For this reason, we did not explicitly restrict the form of the penalty functions in subsection 6.1.2.1 except for the natural assumption of strict increasing monotonicity. Even discontinuous functions with upwards jumps are conceivable.

As we eventually have to commit ourselves to some (naive) approach that takes the resilience of the role categories into account, we finally define the penalty functions as follows:

$$\pi_{LH}(t) := t^3, \quad \pi_{SH}(t) := t^2, \quad \pi_{EF}(t) := t, \quad \pi_{CC}(t) := \sqrt{t}. \tag{6.7}$$

Obviously, as required, the above definitions induce that working time of the lead hand category is penalized to a greater extent than for the secondary help which, in turn, will be spared more than the exterior fire fighting roles and so on. The resulting penalty functions (6.7) are displayed in figure 6.2.

e. Classification of roles into categories

In this paragraph we will classify every role from subsection 6.3.1.1 into one of the categories defined in the last paragraph. The tasks of the lead hand category primarily fall in the field of responsibilities of the first attacking squad. Thus, we set

$$LH := \{AS1\}. \tag{6.8}$$

The first attacking squad is supported in its activities by the second attacking squad and the safety squad. So, these two roles define the secondary help:

$$SH := \{AS2, SS\}. \tag{6.9}$$

All other tasks beyond management and organizing activities, like operating water hoses, driving, or operating the pumper truck, can be accomplished by squadron leaders, water squad, and engineers. The additional action forces MF1 to MF4, that can take miscellaneous roles, also fall in this category. Altogether, this means

$$EF := \{SL1, SL2, WS1, WS2, E1, E2, MF1, MF2, MF3, MF4\}. \tag{6.10}$$

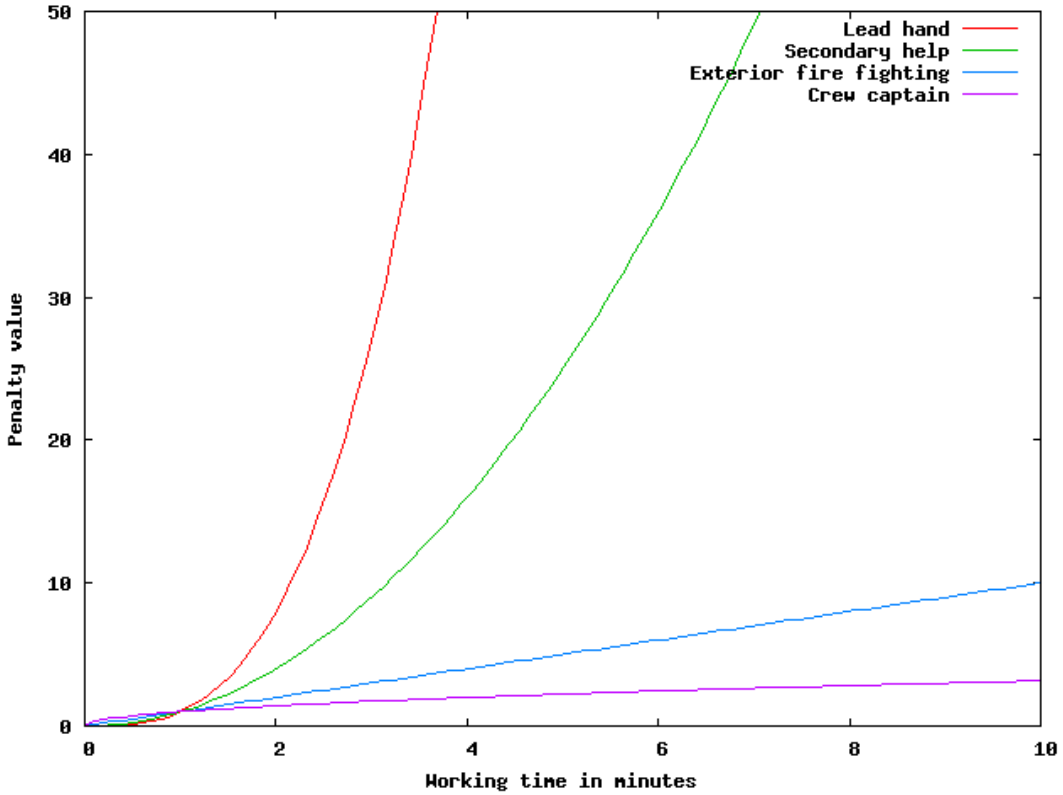


Figure 6.2 Penalty functions for the different role categories

Finally, the managing roles B service, C service, and station officer form the crew captain category:

$$CC := \{BS, CS, SO\}. \tag{6.11}$$

Refering to the objective function F from (6.1), equation (6.7), and the classifications (6.8) to (6.11), the objective function which should be minimized looks as follows for the current setting:

$$\begin{aligned} & F(t_{AS1}, t_{AS2}, \ldots, t_{SO}) \\ & = t_{AS1}^3 \\ & + (t_{AS2}^2 + t_{SS}^2) \\ & + (t_{SL1} + t_{SL2} + t_{WS1} + t_{WS2} + t_{E1} + t_{E2} + t_{MF1} + t_{MF2} + t_{MF3} + t_{MF4}) \\ & + (\sqrt{t_{BS}} + \sqrt{t_{CS}} + \sqrt{t_{SO}}), \end{aligned} \tag{6.12}$$

where $t_{AS1}, t_{AS2}, \ldots$ are the total working times of the roles $AS1, AS2, \ldots$ in a feasible task assignment.

As the construction of the penalty functions is based on the physical and psychic stress a role is exposed to when performing tasks, a penalty value can also be interpreted as stress level. Since the global objective function F is composed of these penalty functions, F can thus be seen as cumulated stress level over all fire brigade roles. This means that a small value of F indicates an assignment of tasks to roles with a balanced workload.

So again, as already stated above and for derivation of penalty functions for the CASASSIGN problem in subsection 4.1.1.2, we just want to motivate the use of our penalty function concept within the given task assignment problem. The exact form and shape of the penalty functions within concrete applications have to be made more precise in subsequent empirical studies on this subject or on related applications like project management with the help of experts (we will give an overwiev on suggestions for prospective studies at the end of section 6.3.3). The same holds for our role category discrimination with respect to the critical body temperature, which is probably just one possible way of many to derive reasonable penalty functions. The crucial point is that neither our formal model from subsection 6.1.2 nor the GWMSA implementation from subsection 6.2 depend on these concrete choices, such that they will still be applicable for different or refined derivations.

6.3.2 Benchmark strategies

We have asked six fire brigade operations managers (Ex1 to Ex6) for their preferred assignment of all 37 subtasks in the context of fighting conflagrations to the available 16 roles listed in subsection 6.3.1.1. We compared the resulting action plan recommendations given by the experts with an assignment that was computed by the GWMSA algorithm.

6.3.3 Simulation results

As already done for the simulations of the other problem classes, for determination of the best combination of input parameters for modified simulated annealing we have again randomly generated 100 of such combinations and applied each of them on 100 randomly generated task allocation problem instances of current type. Finally, the following combination of input parameters showed the best performance on all problem instances:

- Initial temperature $T_{init} = 57.26$.
- Number of runs with constant temperature $\xi = 76$.

- Temperature degression coefficient $\delta = 0.6172$.

Therefore, we have also applied the above combination on our empirical problem instance to compute an optimized assignment of tasks to roles. Besides, we have used $\varepsilon = 0.001$ as termination threshold again.
All computations with the GWMSA algorithm were performed on an Intel (tm) Quad Core with 2.26 GHz and 8 Gigabytes of RAM. The computation time of GWMSA on our empirical problem instance was negligible (less than one second).

The total working times in minutes assigned to each role by the experts and by GWMSA are displayed in table 6.2. The first column contains the shortcuts of the role designations from subsection 6.3.1.1. In the last but one column, the average working time for every role with respect to the expert recommendations is displayed. Finally, the last column contains the working times corresponding to the task allocation computed by GWMSA.
In the last but one row, the total working time sumed up over all roles is computed, and the last row contains the global penalties determined by the value of F.

Role	**Ex1**	**Ex2**	**Ex3**	**Ex4**	**Ex5**	**Ex6**	**Avg**	**GWMSA**
BS	0	1	1	3	5	5	3	5
CS	3	6	1	2	4	1.5	2.92	4.5
SO	6	4	7.5	5	2	5	4.92	3
SL1	3	0.5	2.5	2.5	7.5	1	2.83	1
SL2	0.5	1.5	1	1	0	0.5	0.75	0
AS1	18	9	11	14.5	15	10	12.92	7
AS2	12	2	3	2	5.5	8	5.42	6
WS1	5	8	12	6	6	5.5	7.08	9
WS2	3	0	6	2	2	2.5	2.58	6
SS	4	3	1	1	0	0	2	1
E1	1	2.5	0	2	0	0	0.92	0.5
E2	1	5	1.5	4	4.5	1.5	2.92	0.5
MF1	0	0	0	0	2	2	0.7	0
MF2	0	9	2	2	1	2	2.7	2
MF3	2	0	0	0	0	2	0.7	0
MF4	0	0	0	0	0	2	0.3	0
Sum	58.5	51.5	49.5	47	54.5	48.5	51.58	45.5
F	6011.68	773.95	1370.74	3078.51	3433.90	1088.70	2217.20	409.68

Table 6.2 Working times for each role in minutes

We can notice a clear outperformance of the task allocation computed by GWMSA when considering the main practical objectives as stated in the introduction of section 6.1, namely minimizing total working time and a balanced workload of roles with respect to their stress level:

- The global penalty value of the assignment computed by GWMSA (= 409.68) is clearly less (about 82% on average) than for the action plans recommended by the experts (between 773.95 and 6011.68). As the penalties can be interpreted as measures of stress, the workload of the GWMSA assignment is more balanced with respect to the individual stress level the roles are exposed to. This is as expected, since in contrast to our algorithm, which explicitly minimizes global penalty, the experts have designed their task allocation plans autonomously based on their experience without taking any objective functions into consideration.

- The total processing time of all roles computed by GWMSA of 45.5 minutes is almost 12% less than average processing time of expert recommendations which equals 51.58 minutes. This can be explained by the fact that the algorithm minimizes the global penalty F from equation (6.12) and thus implicitly also minimizes the total processing time, since the penalty functions are strictly monotonically increasing, subject to the working time.

So, altogether we can conclude that our construction of role specific penalty functions in combination with optimization of the corresponding task allocation problem by GWMSA leads to an action plan for fire fighting that is especially suited for meeting practical requirements when fighting major incidents, where experience values are small.
As our problem definition is generic, an application of the task allocation concept on other practical problem instances like project management is conceivable. This could be an interesting topic for prospective studies. Likewise and as already mentioned above, the derivation of the penalty functions for the current application has to be refined with the help of experts. Besides, trying out other optimization techniques like genetic approaches on the given problem instance will be another exercise for future research.

7 Joint simulation

In our joint simulation we have compared the performance and result stability of classic simulated annealing and the GWMSA implementations on the training sets of static CASASSIGN, TESTSUITE, and TASKALLOC when applying different input parameter combinations. We have omitted the online CASASSIGN problem as the involved variables and data sets are very similar to that of the static version with respect to their dimensions, and thus, the results for the static problem definition should generalize to the online version.

7.1 Data setup

For our experiments we have randomly generated 1000 input parameter combinations consisting of initial temperature T_{init}, number of runs with constant temperature ξ and temperature degression coefficient δ. The ranges out of which the input parameters were drawn are shown in table 7.1. We have chosen a greater initial temperature range and a greater range for the number of runs for static CASASSIGN, as its solution space is much bigger than the ones of TESTSUITE and TASKALLOC.

	T_{init}		ξ		δ	
Problem	**lb**	**rb**	**lb**	**rb**	**lb**	**rb**
CASASSIGN	100	10000	100	1000	0.01	0.99
TESTSUITE	1	100	5	150	0.01	0.99
TASKALLOC	1	100	5	150	0.01	0.99

Table 7.1 Ranges of randomly generated input parameters (lb = left bound, rb = right bound)

We applied each parameter combination on the training instances of static CASASSIGN, TESTSUITE, and TASKALLOC from sections 4.3.1, 5.3, and 6.3 by using the classic version of simulated annealing (CSA) from [97] and the corresponding implementations of greedy with modified simulated annealing (GWMSA) for the three problem classes.

7.2 Simulation results

For our implementations of the GWMSA algorithm for static CASASSIGN, TESTSUITE, and TASKALLOC the training results were only used to determine that input parameter combination which performed best on all training instances. This

combination was applied on empirical test instances afterwards. The training results were not analyzed anymore beyond that (except for the static CASASSIGN problem). We will therefore take a closer look at the training results now. In particular, for every randomly generated parameter combination

$$PC_k := (T_{init,k}, \xi_k, \delta_k), \quad k = 1, 2, \ldots, 1000,$$

we first measure the average performance of this parameter triple on each training set

$$TS_j := \{ts_{j,1}, ts_{j,2}, \ldots, ts_{j,|TS_j|}\} \quad \text{for } j = \text{CASASSIGN,TESTSUITE,TASKALLOC}$$

and both algorithms (CSA and GWMSA, respectively). In this context, $ts_{j,l}$ denotes the l.th training instance of problem class j, and $|TS_j|$ is the total number of training instances generated for the j.th problem class. Every training instance, in turn, is a randomly generated instance of the given problem class consisting of solution space $\mathcal{S}$, component set C, objective function F, and evaluation map f, see section 2.1. More formally, we compute

$$\overline{\mathrm{F}}(i, j, PC_k) := \frac{\sum_{l=1}^{|TS_j|} F(S(i, PC_k, ts_{j,l}))}{|TS_j|}$$

for i =CSA,GWMSA, j =CASASSIGN,TESTSUITE,TASKALLOC, and $k = 1, 2, \ldots, 1000$, where $S(i, PC_k, ts_{j,l})$ denotes the solution that is computed by algorithm i with parameter combination PC_k and training instance $ts_{j,l}$ as input. Afterwards, we compute average performance and variance with respect to the input parameter combinations for every problem class and algorithm, namely

$$\overline{\overline{F}}(i, j) := \frac{\sum_{k=1}^{1000} \overline{\mathrm{F}}(i, j, PC_k)}{1000}$$

and

$$Var(\overline{F}(i, j)) := \frac{\sum_{k=1}^{1000} (\overline{\mathrm{F}}(i, j, PC_k) - \overline{\overline{F}}(i, j))^2}{1000}$$

for i =CSA, GWMSA, and j =CASASSIGN,TESTSUITE,TASKALLOC. We furthermore measure the average computation time with respect to all input parameter combinations and training instances,

$$\overline{\mathrm{time}}(i, j) := \frac{\sum_{k=1}^{1000} \sum_{l=1}^{|TS_j|} \mathrm{time}(S(i, PC_k, ts_{j,l}))}{1000 \cdot |TS_j|}$$

for i =CSA, GWMSA, and j =CASASSIGN,TESTSUITE,TASKALLOC, where $\mathrm{time}(S(i, PC_k, ts_{j,l}))$ denotes the computation time in seconds of the applied algorithm i to compute $S(i, PC_k, ts_{j,l})$. The final results are displayed in table 7.2.

j \ i	CSA			GWMSA		
Problem	$\overline{\overline{F}}$	$\mathbf{Var}(\overline{F})$	$\overline{\mathbf{time}}$	$\overline{\overline{F}}$	$\mathbf{Var}(\overline{F})$	$\overline{\mathbf{time}}$
CASASSIGN	$5.4679 \cdot 10^{32}$	$9.6141 \cdot 10^{67}$	36.39	$5.4083 \cdot 10^{9}$	$7.7893 \cdot 10^{19}$	11.06
TESTSUITE	79.7259	9.5148	0.3705	76.2262	0.0061	2.6660
TASKALLOC	132.8090	1220.7300	0.0023	56.8091	0.0255	0.0128

Table 7.2 average objective function value, variance, and average computation time (in seconds) for CSA and GWMSA

We can see that for every problem class the average objective function value $\overline{\overline{F}}$ as well as the variance $\mathrm{Var}(\overline{F})$ is less for GWMSA when compared to classic simulated annealing (CSA). In particular, the variance values of GWMSA amount to less than 0.01% of the classic simulated annealing result variances for every problem class. Thus, the results produced by GWMSA are much more stable than for classic simulated annealing and also better with respect to their quality (measured by the value of the objective function F) on average. Moreover, this stability implies a clearly less solution quality dependence on the choice of the input parameters for GWMSA.
For the average computation time, however, the results are varying between problem classes:
For the very complex CASASSIGN class, GWMSA converges faster than classic simulated annealing (11 seconds compared to 36 seconds for CSA) on average. On the other side, for the problem classes TESTSUITE and TASKALLOC the classic version is multiple times faster than GWMSA. While for TASKALLOC the average computation time of GWMSA is still clearly less than one second and thus negligible, for TESTSUITE the corresponding value of almost 3 seconds lies within a timeframe that is still acceptable for most practical applications.

8 Website ranking problem

8.1 Problem specification

8.1.1 Introduction

In this chapter we consider the problem of ranking websites that match a specific search string. We will refer to this problem in the following sections by just using the abbreviation WEBRANK. Search engines usually order websites with respect to their topic relevance according to some weighting criterion like Google PageRank [26]. A more recent approach from Joachims [86] collects and employs user relevance feedbacks for optimizing search engines as follows: Every time a user searches documents that match a specific search string, the individual ranking of the search results implicitly given by the user is stored and used for subsequent searches. The individual user ranking is derived by assigning a weight to every website the user has clicked on. At this, to be able to distinguish between different websites, they are represented by feature vectors that describe their individual characteristics. The longer a user stays on a website, the higher its weight will be. The idea behind this is that a long retention time usually corresponds to a high relevance of a website with respect to a topic. Hence, this site should be ranked higher than other sites where the user only stayed for a short time period. Besides, if a user clicks on a link displayed in the search results more below, then the corresponding website should be weighted higher than all document links displayed above of the clicked link, presupposing that these were disregarded by the user. In this case we can then conclude that the previous document links were assessed as not so relevant by the user for the current query and thus should be weighted less.
The feature vectors that represent the websites and the corresponding weights are the components of a query. A set of such queries is then used to train a model that is applied to predict a website ranking for later queries. Thus, altogether, the following two components have to be implemented for this approach:

- A strategy for learning a model from the user feedback.
- A classifier that ranks web pages from a search by using the learned model from previous step.

Joachims has used a support vector machine (SVM) strategy to master the two tasks from above. His C implementations $\text{SVM}^{\text{light}}$ and the newer version SVM^{rank} consist of a learning and a classification program that can also be used to solve

several other classification and ranking problems.[1]
In section 8.3 we will empirically compare different strategies based on Joachims's SVM implementations to rank webpages of a query. In particular, these are combinations of the SVM$^{\text{light}}$ algorithm to learn a model and a simple voting strategy to classify a list of websites within a query. Moreover, we will also combine SVM$^{\text{light}}$ with an implementation of our GWMSA strategy and finally compare these two combinations with the use of SVM$^{\text{rank}}$ for both tasks, learning a model as well as classification of websites.
We will see that the predicted website rankings of the three strategies do not differ significantly with respect to Kendall's tau distance. However, SVM$^{\text{rank}}$ is slightly faster than the other two approaches. Before discussing these issues in detail, we will first formally describe the website ranking problem up next.

8.1.2 Formalization

We assume that we are given $|Q|$ *queries*

$$Q := \{q_1, q_2, \ldots, q_{|Q|}\}$$

and $|C|$ different *websites*

$$C := \{c_1, c_2, \ldots, c_{|C|}\},$$

where every website, in turn, is characterized by a set of $|\Phi|$ real valued *features*, whose values depend on the query. Thus, we have a map

$$\Phi \,:\, Q \times C \to \mathbb{R}^{|\Phi|}.$$

We will denote the set of all possible rankings of the websites in C by $\mathcal{S}$ with

$$\mathcal{S} := \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$$

and identify every S_i with a vector of the form

$$S_i := (c_{i_1}, c_{i_2}, \ldots, c_{i_{|C|}})$$

with $i_j \neq i_k$ for every $j, k \in \{1, 2, \ldots, |C|\}, j \neq k$ and $i = 1, 2, \ldots, |\mathcal{S}|$. The "ranking vector" S_i from above is interpreted in a way that website c_{i_1} has the highest ranking within this specific solution, c_{i_2} has the second highest ranking and so on.
Obviously, the set of all possible website rankings $\mathcal{S}$ is isomorphic to the symmetric permutation group of rank $|C|$, $\mathcal{S} \cong \mathscr{S}_{|C|}$. This implies $|\mathcal{S}| = |C|!$, and we can identify every solution $S \in \mathcal{S}$ with a permutation $\sigma \in \mathscr{S}_{|C|}$. We will therefore use

[1] http://svmlight.joachims.org

the terms ranking vector and permutation synonymously in the rest of this chapter. Furthermore, we define a map

$$\theta : Q \to \mathcal{S},$$

that assigns a ranking vector to every query q_i, $i = 1, 2, \ldots, |Q|$. The ranking position of a specific website within this vector corresponds to its individual relevance for the current query. Consequently,

$$\eta : \mathcal{S} \times C \to \{1, 2, \ldots, |C|\}$$

provides this ranking position of a specific website $c \in C$ in a solution vector $S \in \mathcal{S}$.

We suppose that the set of queries is divided into two parts, a training set Q_{train} and a test set Q_{test} with $Q = Q_{train} \cup Q_{test}$ and $Q_{train} \cap Q_{test} = \emptyset$.

The website ranking problem can now be characterized as follows:

- Given the permutations for queries in the set Q_{train}, $\theta|_{Q_{train}}$, find a generalization

$$\tilde{\theta} : Q \to \mathcal{S}$$

with $\tilde{\theta}|_{Q_{train}} = \theta$, such that Kendall's tau distance,

$$\begin{aligned} \mathcal{K}(q) :=& |\{(j,k) \in \{1, 2, \ldots, |C|\} : j < k, \\ & ((\eta(\theta(q), c_j) < \eta(\theta(q), c_k)) \wedge (\eta(\tilde{\theta}(q), c_j) > \eta(\tilde{\theta}(q), c_k))) \vee \\ & ((\eta(\theta(q), c_j) > \eta(\theta(q), c_k)) \wedge (\eta(\tilde{\theta}(q), c_j) < \eta(\tilde{\theta}(q), c_k)))\}|, \end{aligned} \tag{8.1}$$

is minimized for all $q \in Q_{test}$.

The ideal choice would be $\tilde{\theta} := \theta$, of course. However, as we assume $\theta|_{Q_{test}}$ to be unknown, a different approach to determine $\tilde{\theta}$ has to be chosen. Moreover, to be able to implement the GWMSA strategy (but also any other global optimization heuristic) for computation of $\tilde{\theta}$, it goes without saying that an objective function has to be given which can be evaluated for every solution in the solution space. However, when considering the formal descriptions above, this requirement is obviously not satisfied for the problem at hand, since Kendall's tau distance, which serves as objective function for the WEBRANK problem, cannot be evaluated. Before introducing an GWMSA implementation and other benchmark strategies, we will therefore first derive an alternative objective function in the next section.

8.1.3 Objective redefinition

8.1.3.1 Learning a preference function

Cohen et al. [38] suggest to divide the website ranking problem into two parts. In the first step, a preference function

$$\Omega : C \times C \times Q \to [0, 1]$$

of the form

$$\Omega(c_i, c_j, q) := \sum_{l=1}^{|\Phi|} \upsilon_l \cdot \Xi_l(c_i, c_j, q) \tag{8.2}$$

with

$$\Xi_l(c_i, c_j, q) := \begin{cases} 1 & \text{if} \quad \Phi(q, c_i)^T \cdot \epsilon_l > \Phi(q, c_j)^T \cdot \epsilon_l \\ 0 & \text{if} \quad \Phi(q, c_i)^T \cdot \epsilon_l < \Phi(q, c_j)^T \cdot \epsilon_l \\ \frac{1}{2} & \text{otherwise} \end{cases},$$

for $i, j \in \{1, 2, \ldots, |C|\}$ and $q \in Q$, where ϵ_l denotes the l^{th} unity vector in $\mathbb{R}^{|\Phi|}$, is defined. The corresponding weight vector $\vec{v}$ is learned based on the queries in Q_{train}. In [38], an algorithm for weight allocation is introduced. However, we will choose a support vector machine approach to obtain $\vec{v}$, which will be described in the next subsection.

8.1.3.2 Support vector machine approach

In [86], Joachims defines a permutation $\sigma_{\vec{v}} \in \mathscr{S}_{|C|}$ with respect to a weight vector $\vec{v} \in \mathbb{R}^{|\Phi|}$ by

$$(\sigma_{\vec{v}}(q))(i) < (\sigma_{\vec{v}}(q))(j) :\Leftrightarrow \vec{v}^T \cdot \Phi(q, c_i) > \vec{v}^T \cdot \Phi(q, c_j) \tag{8.3}$$

for every $q \in Q$. Let

$$Q_{train} := \{q_1^*, q_2^*, \ldots, q_{|Q_{train}|}^*\}.$$

The weight vector $\vec{v}$ is learned by solving the following ranking support vector machine optimization problem:

- Minimize

$$\Psi(\vec{v}, \vec{\phi}) = \frac{1}{2}\vec{v}^T\vec{v} + \Lambda \cdot \sum_{i=1}^{|C|}\sum_{j=1}^{|C|}\sum_{k=1}^{|Q_{train}|} \phi_{ijk},$$

- such that

$$\forall (i,j) \in \mathcal{I}^2 \,:\, \vec{v}^T\Phi(q_k^*, c_i) \geq \vec{v}^T\Phi(q_k^*, c_j) + 1 - \phi_{i,j,k}, k = 1, 2, \ldots, |Q_{train}|,$$
$$\forall i, j, k \,:\, \phi_{ijk} \geq 0,$$

where $\mathcal{I}^2 := \{(i,j) \in \{1, \ldots, |C|\}^2 \,|\, \eta(\theta(q_k^*), c_i) < \eta(\theta(q_k^*), c_j)\}$.

Λ is a parameter that allows trading-off margin size against training error, and the variables ϕ_{ijk} are non-negative slack variables.
As already mentioned at the beginning in section 8.1.1, Joachims has provided an implementation called SVM$^{\text{light}}$ (and SVM$^{\text{rank}}$, respectively, which is a faster version of SVM$^{\text{light}}$) to solve the above problem. We will also use this implementation to compute the weight vector $\vec{v}$ for the preference function from (8.2).
Finally, we are now able to define the surrogate objective function which will serve as input for GWMSA and another strategy to solve the WEBRANK problem: We use the definition of the preference function Ω from (8.2) to determine a permutation for every $q \in Q_{test}$ by finding $S \in \mathcal{S}$ that maximizes

$$F_{q,\Omega}(S) := \sum_{i,j \in \{1,\ldots,|C|\} \,:\, \eta(S,c_i) < \eta(S,c_j)} \Omega(c_i, c_j, q) \tag{8.4}$$

individually for q and finally setting $\tilde{\theta}(q) := S$.

Cohen et al. show that this optimization problem is $\mathcal{NP}$-complete. Thus, efficient algorithms for computation of an optimal solution most likely will not exist. Therefore, the authors introduce a greedy strategy to compute an approximatively optimal ordering function $\tilde{\theta}$ which orders the websites by their potential. We will also use their greedy algorithm for our GWMSA implementation for the WEBRANK problem, which will be described in section 8.2.1.

For connection of the definitions from sections 2.1 and 8.1.2, we can identify the set of components C with the websites. Furthermore, the objective function is given by the agreement definition (8.4) which is individually maximized for every query $q \in Q_{test}$ and given preference function Ω. We can thus set $F := F_{q,\Omega}$. Finally, the solution space $\mathcal{S}$ corresponds to the set of all possible website rankings.

8.2 Implementation of GWMSA

8.2.1 Greedy

As already mentioned, we will use the greedy algorithm from Cohen et al. [38] for our GWMSA implementation. To be able to describe this algorithm with the notation of section 3.1, we need some additional definitions.

First of all, we define a partial solution S_{part} as a partial ranking of the websites in C, which is identified by a vector of different websites of maximum length $|C|$ (the vectors of length $|C|$ correspond to complete solutions). To be able to complete a partial solution

$$S_{part} := (c_1^*, c_2^*, \ldots, c_{|S_{part}|}^*) \in \mathcal{S}_{part}$$

with $|S_{part}| < |C|$, we define an operation

$$\oplus : \mathcal{S}_{part} \times C \to \mathcal{S}_{part}$$

by

$$(c_1^*, c_2^*, \ldots, c_{|S_{part}|}^*) \oplus c := (c_1^*, c_2^*, \ldots, c_{|S_{part}|}^*, c)$$

for all $c \in C \setminus \{c_1^*, c_2^*, \ldots, c_{|S_{part}|}^*\}$.

Cohen et al. use a greedy approach with potentials as evaluation map f. With the notation from above it can be defined recursively as follows:

$$f_q((), c) := \sum_{\tilde{c} \in C} \Omega(c, \tilde{c}, q) - \sum_{\tilde{c} \in C} \Omega(\tilde{c}, c, q),$$
$$f_q(S_0 \oplus c', c) := f_q(S_0, c') + \Omega(c', c, q) - \Omega(c, c', q),$$

for every partial solution $S_0 \in \mathcal{S}_{part}$ with $|S_0| < (|C| - 1)$ and $c, c' \in C \setminus S_0$ with $c \neq c'$ ("()" denotes an "empty" ranking vector).

By replacing "$\cup$" with our recently defined operator "$\oplus$", Cohen's greedy algorithm has now exactly the structure of the abstract specification from section 3.1.

8.2.2 Modified simulated annealing

Like for the other problem classes considered so far, by using our generalized notation from above it is sufficient to specify the neighborhood of a website ranking and the

candidate generation probability distribution. Analogously to the greedy strategy from last section, the individual steps of the modified simulated annealing algorithm applied on the WEBRANK problem are the same as for the abstract specification from section 3.1.2.3. However, as we now consider a maximization problem (note that the objective functions of all previously considered problem classes had to be minimized), either the two conditions within the inner loop have to be changed accordingly by swapping F_{new} and F_{curr} or, equivalently, $-F$ is used as objective function instead of F.
In any case, at the end, for a given query $q \in Q_{test}$ the corresponding website ranking is defined by setting $\tilde{\theta}(q) := S_{best}$, where S_{best} is the output vector of the modified simulated annealing algorithm.

8.2.2.1 Neighborhood definition

Given a query $q \in Q$ and a ranking vector $S \in \mathcal{S}$, we define the neighborhood of S by all other orderings $S' \in \mathcal{S}$ that only differ in the order of two elements. Formally,

$$N(S) := \{S' \in \mathcal{S} \mid \exists i \in \{1, \ldots, |C| - 1\} \, : \, S' = \sigma_{(i,i+1)}(S)\},$$

where $\sigma_{(i,i+1)}$ is the permutation operator that swaps the i^{th} and $(i+1)^{\text{th}}$ entry of a vector. The above definition obviously implies $|N(S)| = |C| - 1$ for all $S \in \mathcal{S}$.

8.2.2.2 Definition of candidate generation probabilities

We define candidate generation probabilities proportional to the agreement values of the corresponding solutions, since this is our objective function. So, given an ordering $S \in \mathcal{S}$ we define

$$\mathbb{P}_{cg}(S, S') := \begin{cases} \frac{F(S')}{\sum_{i=1}^{|C|-1} F(\sigma_{(i,i+1)}(S))} & \text{if } S' \in N(S), \\ 0 & \text{else.} \end{cases}$$

8.3 Empirical simulation

In our empirical studies we compare the following three strategies to compute a solution for the website ranking problem:

1. Learning a preference function Ω with SVM$^{\text{light}}$ and classification by simple voting.

2. Learning a preference function Ω with SVM$^{\text{light}}$ and classification by greedy with modified simulated annealing.

3. Learning a model and classification with SVM$^{\text{rank}}$.

We will first introduce the datasets used for our simulations in the next subsection and describe the benchmark strategies in detail in subsection 8.3.2. Finally, in subsection 8.3.3 the simulation results are discussed.

8.3.1 Data setup

We have evaluated the three strategies from above on the LETOR datasets for supervised ranking from Microsoft.[2] These consist of two large scale query sets from 2007 and 2008 with thousands of queries. Each of the sets contains a training part Q_{train} for learning a model or a preference function. The second part of queries Q_{test} is used for testing the learned model. The characteristics of the data sets are summarized in table 8.1. $|Q_{train}|$ and $|Q_{test}|$ are the number of queries in the training and in the test data sets, respectively. $|C|$ denotes the number of websites which are to be ordered, and $|\Phi|$ is the number of features that are measured for every query and website.

	LETOR 2007	LETOR 2008
$\|Q_{train}\|$	5076	2352
$\|Q_{test}\|$	1692	784
$\|C\|$	40	20
$\|\Phi\|$	46	46

Table 8.1 Characteristics of the LETOR datasets.

8.3.2 Benchmark strategies

8.3.2.1 Simple voting

Given a preference function Ω of the form (8.2), an ordering function $\tilde{\theta}$ can also be derived by evaluating pairwise website preferences and counting the preferences for every website. This strategy, which is also known as *simple voting*, consists of the following steps:

[2] http://research.microsoft.com/en-us/um/beijing/projects/letor//

Input:

- Set of websites C.
- A query $q \in Q_{test}$.
- Preference function Ω.

Algorithm:

/* initialize vote array: */
for all $i = 1, 2, \ldots, |C|$ **do**
  $\Sigma_i := 0$.
end for
/* Evaluate pairwise preferences: */
for all $i = 1, 2, \ldots, (|C| - 1)$ **do**
  for all $j = (i + 1), \ldots, |C|$ **do**
    if $\Omega(c_i, c_j, q) > 0.5$ **then**
      $\Sigma_i := \Sigma_i + 1$.
    else
      $\Sigma_j := \Sigma_j + 1$.
    end if
  end for
end for
/* Initialize website index list and solution vector: */
$S_0 := ()$.
$\mathcal{I} := \{1, 2, \ldots, |C|\}$.
/* Construct preference order from voting list: */
for all $i = 1, 2, \ldots, |C|$ **do**
  /* Find index of website that has i^{th} highest vote: */
  $k := \mathcal{I}_1$.
  $v_{max} := \Sigma_{\mathcal{I}_1}$.
  for all $j = 2, \ldots, (|C| - i + 1)$ **do**
    if $\Sigma_{\mathcal{I}_j} > v_{max}$ **then**
      $k := \mathcal{I}_j$.
      $v_{max} := \Sigma_{\mathcal{I}_j}$.
    end if
  end for
  $S_0 := S_0 \oplus c_k$.
  $\mathcal{I} := \mathcal{I} \setminus \{k\}$.
end for

Output:

- Permutation $S_0 \in \mathcal{S}$.

Like for our GWMSA implementation described in section 8.2, we have used the SVM$^{\text{light}}$ algorithm from Joachims for learning the preference function Ω (revise section 8.1.3.2 for details about the support vector machine approach) based on the given website rankings of the queries in Q_{train}. Afterwards, we have used this preference function to predict the (unknown) website rankings of the queries in Q_{test} with the simple voting algorithm from above. Just the same as before, the corresponding website ranking for query $q \in Q_{test}$ is defined by finally setting $\tilde{\theta}(q) := S_0$, where S_0 is the output vector of simple voting.

8.3.2.2 Learning and classification by SVM$^{\text{rank}}$

In all previously described approaches, the agreement F from (8.4) is either explicitely maximized by GWMSA or implicitely by simple voting to obtain a website ranking for the queries in Q_{test}. Alternatively, the learned ranking function $\sigma_{\vec{v}}$ from (8.3) in section 8.1.3.2 can also be used directly to classify websites within a given query from Q_{test} by constructing a solution vector $S \in \mathcal{S}$ entry by entry via

$$\eta(S, c_i) := \sigma_{\vec{v}}(q)(i), \quad i = 1, 2, \ldots, |C|.$$

This is exactly the strategy of the classification programs implemented in SVM$^{\text{light}}$ as well as in SVM$^{\text{rank}}$. Consequently, we will use this approach as another benchmark strategy in our simulations, whose results will be presented and discussed in the next subsection.

8.3.3 Simulation results

For both LETOR datasets, we have first learned a model (preference function) by deriving a weight vector based on the queries in Q_{train}. Afterwards, the websites of each query $q \in Q_{test}$ were classified with the learned model by prediction of an ordering $\tilde{\theta}(q)$. Finally, the actual ordering $\theta(q)$ was derived from the given target values in the test set, and Kendall's tau distance $\mathscr{K}(q)$ was computed. The average results with respect to the test set of queries Q_{test} are displayed in table 8.2 (time denotes the average computation time in seconds for one query).

	LETOR 2007		LETOR 2008	
Algorithm	$\overline{\mathscr{K}}$	$\overline{\text{time}}$	$\overline{\mathscr{K}}$	$\overline{\text{time}}$
SVM$^{\text{light}}$+SV	71.7969	0.2447	21.199	0.0982
SVM$^{\text{light}}$+GWMSA	71.789	1.5720	21.2015	0.6569
SVM$^{\text{rank}}$	71.7939	0.0054	21.236	0.0007

Table 8.2 Simulation results (SV = simple voting, GWMSA = greedy with modified simulated annealing).

We can see that the average values of Kendall's tau distance do not differ significantly between the three strategies. Most surprisingly, the very simple structured voting algorithm for classification seems to be able to keep up with the quite more complex support vector machine approach SVM^{rank}. However, with respect to the average computation time, SVM^{rank} as well as $\text{SVM}^{\text{light}}$ with simple voting are significantly faster than the combination of $\text{SVM}^{\text{light}}$ for learning of a preference function and greedy with modified simulated annealing. This is not surprising, since modified simulated annealing is comparatively complex due to the need of a recalculation of the candidate generation probability distribution in every iteration.
Altogether, at least for dataset types like those used in this simulation study, simple approximation algorithms are sufficient to compute good solutions for the website ranking problem. As a consequence, there does not seem a necessity for application of a global optimization strategy like simulated annealing, since no significant result improvement in comparison to local techniques is observable.
Finally, one task for prospective studies are simulations on other data sets to verify the similar performance of the three tested approaches with respect to result quality measured by Kendall's tau distance. Furthermore, additional strategies for learning a preference function should be investigated.

9 Related work

Combinatorial optimization problems and approximation strategies for their solution have been studied extensively over the last decades. The Job Shop Scheduling problem (JSS), which is underlying to the CASASSIGN problem class, was firstly presented by Graham [71] in 1966. For the special case of two machines and an arbitrary number of jobs, this problem can efficiently be solved with Johnson's algorithm [88]. Garey [63] provided a proof in 1976 that JSS is NP-complete for more than two machines. For this general case many different approximation techniques for JSS have been suggested. These can roughly be divided into priority dispatch rules, bottleneck based heuristics, artificial intelligence, local search methods, and meta-heuristics, see [85]. Van Laarhoven et al. were the first ones who applied classic simulated annealing on JSS [135].
The set covering problem, which is a special case of the TESTSUITE and TASKALLOC problem classes, has also been discussed in the literature at large. In [93], Karp has provided a proof that set cover is an $\mathcal{NP}$-complete problem. Subsequently, many different approximation strategies have been suggested and applied to it like greedy [36] and branch and bound [12] as well as iterative search heuristics like classic simulated annealing [29] and genetic algorithms [18]. Finally, the greedy algorithm from [36] was proven to be the best-possible polynomial time approximation algorithm for set cover, see [2].

The classic simulated annealing algorithm, which forms the basis of our metaheuristical approach, was first proposed in [97]. Besides JSS and set cover, it has also been applied on various other combinatorial optimization problems. This includes but is not limited to graph coloring and number partitioning [87] as well as vehicle routing [112], the traveling salesman problem [105], and VLSI design [122]. [131] gives a detailed overview of simulated annealing applications in general, whereas [98] focus on problems from operations research in particular.

Within our metaheuristical approach, a modified version of the simulated annealing algorithm uses the result that was computed by a greedy strategy before as initial solution for further optimization iterations. The idea of combining greedy with a search heuristic has first been suggested in [58]. The authors apply their concept of greedy randomized adaptive search procedure (GRASP) to different combinatorial optimization problems including set cover. As they make use of a local search heuristic, they need multiple iterations to determine the global optimum. However, since the number of iterations has to be limited, detection of the global optimum cannot be guaranteed. This is a clear drawback compared to our approach, as simulated annealing is a global search heuristic and thus, one iteration is usually sufficient to

determine the global optimum.
In [9] the authors use linear programming for computation of an initial solution as starting point for local search. They demonstrate the superiority of their strategy on MAX-CUT and MAX-k-SAT problems compared to iterated local search with random starting point. However, since application of a linear programming approach requires the objective function to be linear, this is not a practicable way to compute good solutions for the optimization problems considered in this work as these are non-linear in general.

Vice versa, there exist different approaches that combine simulated annealing with another local or global search strategy. In [106] the authors restrict the iterative search procedure of simulated annealing to locally optimal solutions by embedding a local search into the classic simulated annealing algorithm. This strategy requires that all locally optimal solutions can be computed very efficiently, of course. However, for more complex optimization problems like our CASASSIGN problem class this approach is impracticable. The reason is that computation of one locally optimal solution might indeed be computationally efficient for such problem types but still too resource-intensive for multiple iterated computations of a local optimization strategy like greedy.
A very similar approach is suggested by [46] where a local optimization algorithm is started every time after a neighbored solution has been generated by simulated annealing. The hybrid algorithm of [116] also falls into this category. Obviously, these strategies share the same problems as the one from [106].

Simulated annealing has also been combined with other global optimization techniques like genetic algorithms [39], [149], tabu search [112], and particle swarm optimization [136]. However, all these combined strategies incorporate the general drawbacks of the respective algorithms that we have discussed in subsection 2.2.7.

Further combinations of simulated annealing with local search or other optimization strategies like [142], [150], [123], and [24] are very problem specific and cannot be generalized offhand for other problem types as we did for GWMSA in chapter 3.

Much research has also been done on the question how to choose the input parameters of classic simulated annealing meaningfully. Concerning the choice of the initial temperature, [134] and [99] have proposed different methods to select the initial temperature based on the initial acceptance ratio. [10] and [133] deal with the cooling schedule and suggest to use logarithmic cooling schedules instead of a constant temperature degression. Finally, [1] have studied the impact of varying the number of iterations. Nevertheless, as already mentioned in chapter 1, due to the dependency on the considered problem type, the question of the optimal input

parameter choice is still open. The most recent strategies are surveyed in [131].

In the remainder of this chapter we want to list related articles about the four problem classes that are considered within this work. Regarding the CASASSIGN problem, public security has gained a lot of interest in recent years. The public funded research projects SOGRO [125] (Instant rescue at big accident with masses of casualties) or SoKNOS [126] (Service-Oriented Architectures Supporting Networks of Public Security) address the optimized care of injured in disasters. SOGRO focuses on techniques to shorten the first chaotic phase immediately after a major incident and before assignment of casualties to transport vehicles and hospitals. On the other side, SoKNOS mainly deals with the adaptation of service oriented architectures, semantics, and human adapted workplaces in public security. Information processing is also a major topic within SoKNOS but especially with a focus on visualization.
Moreover, there exist several works about emergency logistics planning. For example [143] have developed a time-space network model with the objective of minimizing the length of time needed for emergency repair, subject to related operating constraints. This leads to an integer network flow problem with side constraints, and the authors have developed a heuristic to solve it. [102] use genetic algorithms and a vehicle assignment approximation strategy to solve a logistics model for delivery of critical items in a disaster relief operation. [27] deal with the problem of assigning units to tasks but in a more general context. They have constructed a real-time decision support system which uses optimization methods, simulation, and the judgement of the decision maker. Özdamar has extensively studied emergency logistics planning. In his first study about emergency logistics he has developed a hierarchical multi-criteria methodology for helicopter planning during a disaster relief operation in [14]. Some years later in [145] he has focused on the problem of dispatching commodities to distribution centres in emergency situations. The used mathematical model is a hybrid of a multi-commodity network flow problem and a vehicle routing problem. The authors solved it by coupling the sub-models with relaxed arc capacity constraints using Lagrangian relaxation.
An approximate dynamic programming approach for making ambulance redeployment decisions in an emergency medical service system is used in [118]. The authors deal with the question where idle ambulances should be redeployed, to maximize the number of calls reached within a given delay threshold.
The same authors have considered the problem of finding a static deployment of ambulances in an emergency medical service system in [117]. In this context, the goal is to allocate a given number of ambulances among a set of bases to minimize the fraction of calls that are not reached within a time standard.

We have considered two different versions of the CASASSIGN problem in this work: A static problem definition that deals with planning of security measures in the

run-up to a mass event *before* a possible incident and an online definition which considers the problem of assigning casualties to avaiable ambulances and physicians in hospitals *after* a major incident has occurred. Such online problem definitions and strategies to solve them have been developed for various combinatorial optimzation problems. [74] give a survey of some underlying models. The same authors also discuss online optimization of real-world transportation systems and concentrate on those arising in production and manufacturing processes, such as in company internal logistics, see [73].
Different online scheduling problems are the topic of several other studies:
[76] introduce a self-adjusting dynamic scheduling class of algorithms to schedule independent tasks on the processors of a multiprocessor system. [61] develop and experimentally compare policies for the control of a system of multiple elevators with capacity one in a transport environment with multiple floors. They show that a reoptimization policy for minimizing average squared waiting times can be implemented to run in real-time using dynamic column generation. A novel approach which tries to approximate the regret of a decision in the context of an online stochastic optimization problem is introduced in [22]. The authors apply their regret algorithm on the problem of online packet scheduling in networks as well as on the online multiple vehicle routing problem with time windows. Vehicle routing is also the topic of [11] who deal with online traffic engineering and present a new routing scheme referred to as least interference optimization (LIO), where the online routing process uses the current bandwidth availability and the traffic flow distribution to achieve traffic engineering in IP networks. A natural online version of linear optimization, which can be applied on several combinatorial problems like max-cut, variants of clustering, and the classic online binary search tree problem, is considered by [90].

The problem of controlling the size of a test suite was addressed for the first time in [77], where also the equivalence to the set covering problem was stated. Afterwards, several other works regarding test suite reduction were published like [111] and [32], where the general test suite reduction activity is described. [89] consider test suite reduction for modified condition and decision coverage, and in [79] the implications of test suite reductions for testing and effects on the test quality were investigated. Empirical results are reported in [121], [148], and [129]. In recent years also multi-objective test suite optimization problems have been considered as in [146].

With respect to the task allocation problem one of the first works regarding this topic is [34], where the authors deal with the problem of balancing out the interprocessor overhead in distributed systems. Most of the subsequent related studies, such as [113], [13], [33], and [35] focus on organizing the allocation of tasks in distributed computing environments.

Based on the consideration of distributed environments, a related application of the task allocation problem was identified in [31], where the problem of assigning tasks in a multi-robot environment was addressed for the first time. The first solution approaches were provided in [107] and [103]. Post-millenial, after robot technology was more well-engineered, a countless number of studies about multi-robot task allocation were published. Recent works include [42] who provide a new formalization of the multi-robot task allocation problem by vacancy chain scheduling and [23] who present a scalable approach by defining stochastic control policies to dynamically allocate a swarm of homogenous robots to multiple tasks. A survey on distributed autonomous robotic systems in general, including the task allocation problem is given by [4].
Also around the turn of the millenium the third big field of applications was found, namely task allocation in connection with the coordination of unmanned aerial vehicles (UAVs), see [20]. The latest publications in this area are [83] who analyze task assignment for heterogenous air vehicles using a guaranteed conflict-free assignment algorithm and [100], where a distributed heuristic search algorithm for allocating the individual tasks in a task specification tree is introduced.
Besides, many other applications where task allocation is relevant were analyzed in the last years, for example in social networks [137] and in economics [44].
Classic simulated annealing has been applied on the task allocation problem for parallel computing for the first time in [57]. Subsequently, in [132] simulated annealing was used to allocate hard real-time tasks, and [138] combined simulated annealing with a list-based heuristic for task allocation. Further related applications are introduced in [19] and [6].

The WEBRANK problem, which has been considered last, belongs to the general class of object ranking problems, that have already been widely discussed in the literature. Yu et al. study the ranking aggregation problem in distributed systems by proposing three efficient algorithms that take data distributions into account in [147]. In [110], a domain-independent object-level link analysis model to rank objects within a specific domain is introduced. This is done to regard the fact that traditional page rank models are no longer valid for object popularity calculation due to the existence of heterogenous relationships between objects. Freund et al. describe and analyze an algorithm called RankBoost for combining preference functions based on the boosting approach to machine learning in [60]. The Expected Rank Regression method from [91] is used to learn a function with standard regression technique to estimate expected ranks of objects that were derived before.
Besides Joachims's $\text{SVM}^{\text{light}}$ and SVM^{rank} algorithms, which we used in our simulation studies, there exist two other strategies that are based on support vector machines: In [81], a vector space based method is introduced that performs a linear mapping from documents to scalar utility values to guarantee transitivity of the

learned preference relation. The OrderSVM approach, which is discussed in [94], is an algorithm for learning the order of a group of items. While OrderSVM is designed to discriminate whether or not a given object is ranked higher than a specific position, the approach from Herbrich et al. in [81] is able to judge which of two objects precedes the other.
Finally, we refer to [62] for a general overview on object ranking techniques.

10 Conclusion

In this work we have introduced a new metaheuristical approach that combines a greedy strategy with a modified version of the simulated annealing algorithm. In contrast to classic simulated annealing, where the initial solution is randomly generated, the modified version introduced in this work uses the result of a greedy strategy as initial solution for subsequent simulated annealing iterations. Furthermore, within modified simulated annealing, the candidate generation probabilities for choosing a neighbor solution candidate are adjusted in a way that neighbors that more probably lead to good solutions with respect to the optimality criterion are preferred. We have shown how modified candidate generation probabilities can be derived generally from an evaluation map that evaluates extensions of partial solutions of a combinatorial optimization problem. Such an evaluation map can usually be directly deduced from an existing greedy calculus. The resulting abstract algorithm specification can be used for the implementation of the metaheuristic on various combinatorial optimization problems. We have exemplarily demonstrated this procedure for the classic set covering problem.
We have also implemented our metaheuristic for four representative and more complex problem classes. The first problem considers the assignment of casualties to available ambulances and qualified physicians in hospitals after a major incident and is related to a coupled job shop scheduling problem. The second problem deals with extended test suite reduction, where a subset of test cases has to be chosen, such that all functions of a software program are called at least once. However, the total number of function calls has to be minimized and, moreover, the distribution of the function calls should be as balanced as possible. The resulting problem is a generalization of the set covering problem and thus $\mathcal{NP}$-complete. Another generalization of set cover is the problem of task allocation within fire fighting, that we considered as third problem class. Here, a list of tasks has to be assigned to qualified fire brigade roles, such that the resulting assignment is balanced with respect to the workload of the different roles. The last problem class deals with the task of predicting the ranking of websites within search queries. In recent approaches this is done by learning a website preference function from user relevance feedback given by a set of training queries, first. Afterwards, for new queries, we constructed corresponding rankings by maximizing the agreement with the learned preference function.
In one part of our simulation studies we showed the outperformance of our metaheuristic on empirical test data compared to other approximation strategies that are currently used in practice to compute solutions of the first three problem classes introduced above. For the website ranking problem, however, no significant improvements could be achieved in comparison to simple local optimization strategies.

Since another global optimization heuristic that was used as benchmark strategy was also unable to compute better results, we can conclude that for problems of this specific type (which are nevertheless regarded as very complex), the application of global techniques is not favorable in general.

In comparison to other optimization techniques like genetic algorithms, simulated annealing needs less input parameters, and a smaller number of additional operators has to be defined. Besides, simulated annealing processes only one solution at a time, which is just slightly modified within every iteration. If the generation and evaluation of solutions is complex, this is more resource-efficient than for other heuristics where many solutions have to be regenerated and evaluated in every iteration. Furthermore, the simulated annealing algorithm does not put high requirements on the objective function that is to be minimized or maximized with respect to analytical characteristics. It is also applicable on most optimization problems and is very easy to implement. However, one crucial drawback is the high dependence of the solution quality on the choice of the input parameters consisting of initial temperature value, annealing schedule and number of iterations with constant temperature.
In our joint simulation study, we have compared the performance of our new metaheuristic with that of classic simulated annealing by using randomly generated input parameter combinations on problem instances of the first three problem classes from above. The results computed by the metaheuristical approach showed up to be much more stable with respect to the choice of the input parameters than for the classic algorithm. Thus, even though we were unable to answer the open question how to optimally choose the input parameters in this work, we could at least show a way how the dependence on this choice can be decreased significantly.
Implementation and simulation of this metaheuristical approach for other combinatorial but also for continuous optimization problems to validate the results of this work could be an interesting topic for prospective studies. Furthermore, the theoretical background of the metaheuristical approach should be investigated: For the proof of convergence towards the global optimum of the classic simulated annealing algorithm, the reversibility of the induced Markov chain is utilized [72]. However, since the candidate generation probabilities of the modified simulated annealing algorithm are not symmetric, the reversibility of the induced Markov chain does not hold anymore. Therefore, it should be analyzed whether this reversibility is a necessary condition for the convergence of simulated annealing towards the global optimum or whether this requirement can be overridden.

References

1 Aarts, E. and Korst, J. (1989). *Simulated Annealing and Boltzmann Machine*. Chichester: Wiley.

2 Alon, N., Moshkovitz, D. and Safra, S. (2006). Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms, 2*(2), 153-177.

3 Arora, S. and Barak, B. (2009). *Computational Complexity: A Modern Approach*. 1st edition Cambridge University Press.

4 Asama, H., Kurokawa, H., Ota, J. and Sekiyama, K. (2009). *Distributed Autonomous Robotic Systems 8*. Springer.

5 Aschoff, J., Günther, B. and Kramer, K. (1971). *Energiehaushalt und Temperaturregulation*. Urban & Schwarzenberg.

6 Attiya, G. and Hamam, Y. (2006). Task allocation for maximizing reliability of distributed systems: A simulated annealing approach. *J. of Parallel and Distributed Computing, 66*(10), 1259-1266.

7 Ausschuss Feuerwehrangelegenheiten Katastrophenschutz und zivile Verteidigung (AFKzV) (2005). Atemschutz. *Feuerwehr-Dienstvorschrift*.

8 Ausschuss Feuerwehrangelegenheiten Katastrophenschutz und zivile Verteidigung (AFKzV) (2008). Einheiten im Lösch- und Hilfeleistungseinsatz. *Feuerwehr-Dienstvorschrift*.

9 Avdil, A. and Weihe, K. (2009). Local search starting from an LP solution: Fast and quite good. *Journal of Experimental Algorithms, 14*(6).

10 Azencott, R. (1992). *Sequential simulated annealing: speed of convergence and acceleration techniques*. New York: Wiley.

11 Bagula, A., Botha, M. and Krzesinski, A. (2004). Online traffic engineering: the least interference optimization algorithm. *IEEE International Conference on Communications*.

12 Balas, E. and Carrera, M. (1995). A dynamic subgradient-based branch and bound procedure for set covering. *Operations Research, 44*(6), 875-890.

13 Bannister, J. and Trivedi, K. (1983). Task allocation in fault-tolerant distributed systems. *Acta Informatica, 20*(3), 261-281.

14 Barbarosoglu, G., Özdamar, L. and Cevik, A. (2002). An Interactive Approach for Hierarchical Analysis of Helicopter Logistics in Disaster Relief Operations. *European J. of OR, 140*.

15 Barnard, R. and Duncan, H. (1975). Heart rate and ECG response of fire fighters. *Journal of Occupational Medicine, 17*, 247 - 250.

16 BBC News (2000). The Heysel disaster. .

17 Beasley, D., Bull, D. and Martin, R. (1993). An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing, 15*(2).

18 Beasley, J. and Chu, P. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research, 94*(2), 392i-404.
19 Beck, J. and Siewiorek, D. (1996). Simulated Annealing Applied to Multicomputer Task Allocation and Processor Specification. *Proc. of the 8th Symposium on Parallel and Distributed Processing*.
20 Bellingham, J., Tillerson, M., Richards, A. and How, J. (2001). Multi-Task Allocation and Path Planning for Cooperating UAVs. *Proc. of the Conf. on Coordination, Control and Optimization*.
21 Bennett, B., Hagan, R., Banta, G. and Wiliams, F. (1994). *Physiological Responses during shipboard firefighting*. Naval Heatlh Reasearch Center.
22 Bent, R. and Hentenryck, P. V. (2004). Regrets only! online stochastic optimization under time constraints. In *Proceedings of the 19th AAAI* .
23 Berman, S., Halasz, A., Hsieh, M. and Kumar, V. (2009). Optimized Stochastic Policies for Task Allocation in Swarms of Robots. *Transactions on Robotics, 25*(4), 927-937.
24 Blower, P., Fligner, M., Verducci, J. and Bjoraker, J. (2002). On Combining Recursive Partitioning and Simulated Annealing To Detect Groups of Biologically Active Compounds. *Journal of Chemical Information and Modeling, 42*(2), 393-404.
25 Bos, J., Mol, E., Visser, B. and Frings-Dresen, M. (2004). The physical demands upon (Dutch) firefighters in relation to the maximum acceptable energetic workload. *Ergonomics, 47*, 446 - 460.
26 Brin, S. and Page, L. (1998). The anatomy of large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems, 30*, 107-117.
27 Brown, G. and Vassiliou, A. (2006). Optimizing Disaster Relief: Real-Time Operational and Tactical Decision Support. *Naval Research Logistics, 40*.
28 Brown, J. and Stickford, J. (2008). Physiological Stress Associated with Structural Firefighting Observed in Professional Firefighters. Technical Report Blomington Indiana: Indiana University Firefighter Health and Safety Research; School of Health, Physical Education & Recreation; Department of Kinesiology.
29 Brusco, M., Jacobs, L. and Thompson, G. (1999). A morphing procedure to supplement a simulated annealing heuristic for cost and coverage correlated set covering problems. *Annals of Operations Research, 86*(0), 611-627.
30 Bundesministerium der Justiz (1998). Fünftes Sozialgesetzbuch (SGB V): Paragraph 137. .
31 Caloud, P., Wonyun, C., Latombe, J., Le Pape, C. and Yim, M. (1990). Indoor automation with many mobile robots. *Proc. of the Int. Workshop on Intelligent Robots and Systems, 1*, 67-72.
32 Chen, T. Y. and Lau, M. F. (1996). Dividing strategies for the optimization of a test suite. *Information Processing Letters, 60*(3), 135 - 141.

33 Chien-Chung, S. and Wen-Hsiang, T. (1985). A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion. *Transactions on Computers, C-34*(3), 197-203.
34 Chu, W., Holloway, L., Min-Tsung, L. and Efe, K. (1980). Task Allocation in Distributed Data Processing. *Computer, 13*(11), 57-69.
35 Chu, W. and Lan, L. (1987). Task Allocation and Precedence Relations for Distributed Real-Time Systems. *Transactions on Computers, C-36*(6), 667-679.
36 Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of Operations Research, 4*(3), 233-235.
37 Cian, C., Koulmann, N., Barraud, P., Raphael, C. and Jiminez, C. et al. (2000). Influence of variations in body hydration on cognitive function: effect of hyperhydration, heat stress and exercise-induced dehydration. *American Journal of Psychology*, pp. 29 - 36.
38 Cohen, W., Schapire, R. and Singer, Y. (1999). Learning to Order Things. *Journal of Artificial Intelligence Research, 10*, 243-270.
39 Cordon, O., Moya, F. and Zarco, C. (2002). A new evolutionary algorithm combining simulated annealing and genetic programming for relevance feedback in fuzzy information retrieval systems. *Soft Computing, 6*, 308-319.
40 Cormen, T., Leiserson, C. and Rivest, R. (1990). *Introduction to Algorithms*. MIT press.
41 Crespin, U. and Neff, G. (2000). *Handbuch der Sichtung*. Edewecht, Germany: Stumpf & Kossendey-Verlag.
42 Dahl, T., Mataric, M. and Sukhatme, G. (2009). Multi-robot task allocation through vacancy chain scheduling. *Robotics and Autonomous Systems, 57*(6), 674-687.
43 Dantzig, G. (1966). *Linear Programming and Extensions*. Princeton University Press.
44 Dash, R., Vytelingum, P., Rogers, A., David, E. and Jennings, N. (2007). Market-Based Task Allocation Mechanisms for Limited-Capacity Suppliers. *Transactions on Systems, Man and Cybernetics, 37*(3).
45 Der Magistrat der Stadt Frankfurt am Main (2006). Medizinisches Schutzkonzept im Rahmen der Fussballweltmeisterschaft 2006. .
46 Desai, R. and Patil, R. (1996). SALO: Combining Simulated Annealing and Local Optimization for Efficient Global Optimization. *Proc. of the 9th Florida AI Research Symposium*.
47 DFL Deutsche Fussball Liga GmbH (2010). Bundesliga 2010: Die wirtschaftliche Situation im Lizenzfussball. .
48 Donati, A., Montemanni, R., Casagrande, N., Rizzoli, A. and Gambardella, L. (2008). Time Dependent Vehicle Routing Problem with a Multi Ant Colony System. *European Journal of Operational Research, 185*(3), 1174-1191.

49 Dorigo, M. (1992). Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano.*
50 Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science, 344*, 243-278.
51 Dorigo, M. and Di Caro, G. (1999). The Ant Colony Optimization meta-heuristic. In *New Ideas in Optimization*, pages 11-32. McGraw Hill, London.
52 Dorigo, M. and Gambardella, L. (1996). Ant colonies for the travelling salesman problem. *Bio Systems, 43*, 73-81.
53 Duncan, H. and Barnard, R. (1979). Physiological responses of men working in fire fighting equipment in the heat. *Ergonomics, 22*, 521 - 527.
54 Eberhart, R. and Shi, Y. (1998). Comparison between Genetic Algorithms and Particle Swarm Optimization. *Proc. of the 7th Int. Conf. on Evolutionary Programming.*
55 Eberhart, R. and Shi, Y. (2001). Particle Swarm Optimization: Developments, Applications and Resources. *Proc. of the 2001 Congress on Evolutionary Computation.*
56 Elsner, K. and Kolkhorst, F. (2008). Metabolic demands of simulated firefighting tasks. *Ergonomics, 51*, 1418 - 1425.
57 Ercal, F. (1988). Heuristic approaches to task allocation for parallel computing. *Ph.D. Thesis.*
58 Feo, T. and Resende, M. (1995). Greedy randomized adaptive search procedures. *J. of Global Optimization, 6*, 109-133.
59 Fogarty, A., Armstrong, K., Gordon, C., Groeller, H. and Woods, B. et al. (2005). Physiological consequences of wearing personal protective equipment: clothing and helmets. *Environmental Ergonomics*, pp. 383 - 388.
60 Freund, Y., Iyer, R., Schapire, R. and Singer, Y. (1998). An efficient boosting algorithm for combining preferences. *Proc. of the 15th Int. Conf. on Machine Learning*, pp. 170-178.
61 Friese, P. and Rambau, J. (2006). Online-optimization of multi-elevator transport systems with reoptimization algorithms based on set-partitioning models. *Discrete Applied Mathematics, 154* (13), 1908-1931.
62 Fürnkranz, J. and Hüllermeier, E. (2010). *Preference Learning.* Springer-Verlag Berlin Heidelberg.
63 Garey, M. R. (1990). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research, 1*, 117-129.
64 Gass, S. (2010). *Linear Programming: Methods and Applications.* 5th edition Dover Publications.
65 Gendreau, M. and Potvin, J. (2010). *Handbook of Metaheuristics.* 2nd edition Springer.
66 Gledhill, N. and Jamnik, V. (1992). Characterization of the physical demands of firefighting. *Canadian Journal of Sport Science, 17*, 207 - 213.

67 Glover, F. (1989). Tabu Search - Part I. *ORSA Journal on Computing, 1*(2), 190-206.
68 Glover, F. (1990). Tabu Search - Part II. *ORSA Journal on Computing, 2*(1), 4-32.
69 Glover, F. and McMillan, C. (1986). The general employee scheduling problem: An integration of MS and AI. *Computers and Operations Research*.
70 Glover, F., Taillard, E. and de Werra, D. (1993). A user's guide to tabu search. *Annals of Operations Research, 41*, 3-28.
71 Graham, R. (1966). Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal, 54*, 1563-1581.
72 Granville, V., Krivanek, M. and Rasson, J.-P. (1994). Simulated annealing: a proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 16*(6), 652-656.
73 Grötschel, M., Krumke, S. and Rambau, J. (2001b). Combinatorial Online Optimization in Real Time. Technical Report Konrad-Zuse-Zentrum für Informationstechnik Berlin.
74 Grötschel, M., Krumke, S. and Rambau, J. (2001a). Online Optimization of Complex Transportation Systems. Technical Report Konrad-Zuse-Zentrum für Informationstechnik Berlin.
75 Guidotti, T. (1992). Human factors in firefighting: ergonomic, cardiopulmonary, and psychogenic stressrelated issues. *International Archive of Occupational and Environmental Health, 64*, 1 - 12.
76 Hamidzadeh, B., Lau, Y. and Lilja, D. (2000). Dynamic task scheduling using online optimization. *IEEE Transactions on Parallel and Distributed Systems, 11*(11), 1151-1163.
77 Harrold, M., Gupta, R. and Soffa, M. (1993). A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology, 2*(3), 270-285.
78 Haupt, R. and Haupt, S. (2004). *Practical Genetic Algorithms*. John Wiley & Sons.
79 Heimdahl, M. and George, D. (2004). Test-suite reduction for model based tests: effects on test quality and implications for testing. *Proc. of the 19th Int. Conf. on Automated Software Engineering*, pp. 176-185.
80 Henderson, D., Jacobson, S. and Johnson, A. (2003). The Theory and Practice of Simulated Annealing. *Int. Series in Operations Research & Management Science, 57*, 287-319.
81 Herbrich, R., Graepel, T., Bollmann-Sdorra, P. and Obermayer, K. (1998). Learning preference relations for information retrieval. *ICML-98 Workshop: Text Categorization and Machine Learning*, pp. 80-84.
82 Holland, J. (1975). Adaption in Natural and Artificial Systems. *University of Michigan Press.*.

83 How, J., Bertuccelli, L., Choi, H. and Cho, P. (2009). Real-Time Multi-UAV Task Assignment in Dynamic and Uncertain Environments. *Proc. of the AIAA Guidance, Navigation and Control Conference.*

84 Ilmarinen, R. and Mäkinen, H. (1992). Heat strain in firefighting drills. *Proceedings of The Fifth Int. Conference on Environmental Ergonomics*, pp. 90 - 91.

85 Jain, A. S. and Meeran, S. (1999). A State-of-the-art Review of Job-Shop Scheduling Techniques. *European Journal of Operations Research, 113*, 390-434.

86 Joachims, T. (2002). Optimizing search engines using clickthrough data. *Proc. of the 8th ACM Int. Conf. on Knowledge Discovery and Data Mining.*

87 Johnson, D., Aragon, C., McGeoch, L. and Schevon, C. (1991). Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Operations Research, 39*(3).

88 Johnson, S. M. (1994). Optimal Two- and Three-Stage Production Schedules with Set-Up Times Included. *Naval Research Logistics Quarterly, 1*, 61-68.

89 Jones, J. and Harrold, M. (2003). Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE Transactions on Software Engineering, 29*, 195-209.

90 Kalai, A. and Vempala, S. (2002). Geometric algorithms for online optimization. In *Journal of Computer and System Sciences*, pages 26-40.

91 Kamishima, T., Kazawa, H. and Akaho, S. (2005). Supervised Ordering - an empirical survey. *Proc. of the 5th IEE Int. Conf. on Data Mining*, pp. 673-676.

92 Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Proc. of the 16th ACM Symposium on Theory of Computing*, pp. 302-311.

93 Karp, R. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations.*

94 Kazawa, H., Hirao, T. and Maeda, E. (2005). A kernel method for order learning based on generalized order statistics. *Journal of System Computation, 36*(1), 35-43.

95 Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *Proc. of the IEEE Int. Conf. on Neural Networks*, pp. 1942-1948.

96 Kenney, W., DeGroot, D. and Holowatz, L. (2004). Extremes of human heat tolerance: life at the precipice of thermoregulatory failure. *Journal of Thermal Biology, 29*, 479 - 485.

97 Kirkpatrick, S., Gelatt, C. and Vecchi, M. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671-680.

98 Koulamas, C., Antony, S. R. and Jaen, R. (1994b). A Survey of Simulated Annealing Applications to Operations Research Problems. *Omega Journal, 22*, 41-56.

99 Kouvelis, P. and Chiang, W. (1992). A simulated annealing procedure for single row layout problems in flexible manufacturing systems. *International Journal of Production Research, 30*, 717-732.
100 Landen, D., Heintz, F. and Doherty, P. (2010). Complex Task Allocation in Mixed-Initiative Delegation: A UAV Case Study. *Proc. of the 13th Int. Conf. on Principles and Practice of Multi-Agent Systems*.
101 Lemon, P. and Hermiston, R. (1977). The human energy costs of firefighting. *Journal of Occupational Medicine, 19*, 558 - 562.
102 Lin, Y.-H., Batta, R. and Rogerson, P. (2009). A Logistics Model for Delivery of Critical Items in a Disaster Relief Operation: Heuristic Approaches. Working Paper.
103 Lueth, T. and Laengle, T. (1994). Task description, decomposition, and allocation in a distributed autonomous multi-agent robot system. *Proc. of the 3rd Int. Conf. on Intelligent Robots and Systems, 3*, 1516-1523.
104 Luke, S. (2009). *Essentials of Metaheuristics*. Lulu.
105 Malek, M., Guruswamy, M., Pandya, M. and Owens, H. (1989). Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research, 21*(1), 59-84.
106 Martin, O. and Otto, S. (1996). Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research, 63*(1), 57-75.
107 Mataric, M. (1992). Minimizing complexity in controlling a mobile robot population. *Proc. of the Int. Conf. on Robotics and Automation, 1*, 830-835.
108 Murty, K. (1983). *Linear Programming*. 1st edition John Wiley & Sons.
109 Nesterov, Y. and Nemirovskii, A. (1995). *Interior-Point Polynomial Algorithms in Convex Programming*. 1st edition Society for Industrial Mathematics.
110 Nie, Z., Zhang, Y., Wen, J. and Ma, W. (2005). Object-level ranking: bringing order to web objects. In *Proceedings of the 14th international conference on World Wide Web*, pages 567-574.
111 Offutt, A., Pan, J. and Voas, J. (1995). Procedures for reducing the size of coverage-based test sets. *Proc. of the 12th. Int. Conf. on Testing Computer Software*, pp. 111-123.
112 Osman, I. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research, 41*(4), 421-451.
113 Perng-Yi, R., Lee, E. and Tsuchiya, M. (1982). A Task Allocation Model for Distributed Computing Systems. *Transactions on Computers, C-31*(1), 41-47.
114 Poli, R., Kennedy, J. and Blackwell, T. (2007). Particle Swarm Optimization. An overview. *Swarm Intelligence, 1*(1), 33-57.
115 Pukelsheim, F. (2007). Seat bias formulas in proportional representation systems. *Proc. of the 4th ecpr General Conference*.

116 Purushothama, G. and Jenkins, L. (2003). Simulated annealing with local search - a hybrid algorithm for unit commitment. *IEEE Transactions on Power Systems, 18*(1), 273-278.
117 Restrepo, M., Henderson, S. and Topaloglu, H. (2009). Erlang Loss Models for the static Deployment of Ambulances. *Health Care Management Science, 12*(1), 67-79.
118 Restrepo, M., Henderson, S. and Topaloglu, H. (2010). Approximate Dynamic Programming for Ambulance Redeployment. *INFORMS Journal on Computing, 22*, 266-281.
119 Romet, T. and Frim, J. (1987). Physiological responses to fire fighting activities. *European Journal of Applied Physiology, 56*, 633 - 638.
120 Rossi, R. (2003). Fire fighting and its influence on the body. *Ergonomics, 46*, 1017 - 1033.
121 Rothermel, G., Harrold, M., Ronne, J. and Hong, C. (2002). Empirical studies of test-suite reduction. *Software Testing, Verification and Reliability, 12*(4), 219-249.
122 Sechen, C. (1988). *VLSI placement and global routing using simulated annealing*. Kluwer Academic Publishers.
123 Simkin, J. and Trowbridge, C. (1992). Optimizing electromagnetic devices combining direct search methods with simulated annealing. *IEEE Transactions on Magnetics, 28*(2), 1545-1548.
124 Smith, S. (1980). A learning system based on genetic adaptive algorithms. *PhD Thesis, University of Pittsburgh*.
125 SOGRO (2009). Sofortrettung bei Grossunfall, http://www.sogro.de/. .
126 SoKNOS (2009). Service-orientierte Architekturen zur Unterstützung von Netzwerken im Rahmen öffentlicher Sicherheit, http://www.soknos.de/. .
127 Sothman, M., Saupe, K. and Jasenof, D. (1990). Advancing age and the cardiorespiratory stress of fire suppression: determining the minimal standards for cardiorespiratory fitness. *Journal of Human Performance in Extreme Environments, 3*, 217 - 236.
128 Sothman, M., Saupe, K., Jasenof, D. and Blaney, J. (1992). Heart rate response of firefighters to actual emergencies; implications of cardiorespiratory fitness. *Journal of Occupational Medicine, 34*, 797 - 800.
129 Sprenkle, S., Sreedevi, S., Gibson, E., Pollock, L. and Souter, A. (2005). An empirical comparison of test suite reduction techniques for user-session-based testing of web applications. *Proc. of the 21st IEEE Int. Conf. on Software Maintenance*.
130 Staatsanwaltschaft Duisburg (2010). Loveparade - Gemeinsame Presseerklärung von Staatsanwaltschaft und Polizei. .

131 Suman, B. and Kumar, P. (2006). A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society, 57*, 1143-1160.
132 Tindell, K., Burns, A. and Wellings, A. (1992). Allocating hard real-time tasks: An NP-Hard problem made easy. *Real-Time Systems, 4*(2), 145-165.
133 Triki, E., Collette, Y. and Siarry, P. (2004). A theoretical study on the behaviour of simulated annealing to a new cooling schedule. *European Journal of Operational Research, 166*, 77-92.
134 van Laarhoven, P., Aarts, E., van Lint, J. and Willie, L. (1988). New upper bounds for the football pool problem for 6, 7 and 8 matches. *Journal of Combinatorial Theory, 52*, 304-312.
135 van Laarhoven, P. J. M., Aarts, E. and Lenstra, J. (1992). Job Shop Scheduling by Simulated Annealing. *Operations Research, 40*, 113-125.
136 Wang, X. (2004). Hybrid particle swarm optimization with simulated annealing. *Proc. of the Int. Conf. on Machine Learning and Cybernetics, 4*, 2402-2405.
137 Weerdt, M., Zhang, Y. and Klos, T. (2007). Distributed task allocation in social networks. *Proc. of the 6th Int. Conf. on Autonomous Agents and Multiagent Systems*.
138 Wells, B. and Caroll, C. (1993). An augmented approach to task allocation: combining simulated annealing with list-based heuristics. *Proc. of the Euromicro Workshop on Parallel and Distributed Processing*.
139 Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing, 4*, 65-85.
140 Windle, J., Mackway-Jones, K. and Marsden, J. (2006). *Emergency Triage*. Cambridge, England: Blackwell Publishers.
141 Wyndham, C., Strydom, N., Morrison, J., Williams, C. and Bredell, G. et al. (1965). Criteria for physiological limits for work in heat. *Journal of Applied Physiology, 20*, 37 - 45.
142 Yamada, T. and Nakano, R. (1995). Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search. .
143 Yan, S. and Shih, Y.-L. (2007). A Time-Space Network Model for Work Team Scheduling after a Major Disaster. *Journal of the Chinese Institute of Engineers, 30*, 63-75.
144 Yaseen, S. and Slamy, N. (2008). Ant Colony Optimization. *Int. J. of Computer Science and Network Security, 8*(6).
145 Yi, W. and Özdamar, L. (2007). A Dynamic Logistics Coordination Model for Evacuation and Support in Disaster Response Activities. *European Journal of Operational Research, 179*.
146 Yoo, S. and Harman, M. (2007). Pareto efficient multi-objective test case selection. *Proc. of the Int. Symp. on Software Testing and Analysis*.

147 Yu, H., Li, H., Wu, P., Agrawal, D. and El Abbadi, A. (2005). Efficient processing of distributed top-k queries. In Andersen, K., Debenham, J. and Wagner, R., editors, *Database and Expert Systems Applications*, number 3588 in Lecture Notes in Computer Science, pages 65-74. Springer Berlin / Heidelberg.

148 Yu, Y., Jones, J. and Harrold, M. (2008). An empirical study of the effects of test-suite reduction on fault localization. *Proc. of the 30th Int. Conf. on Software Engineering*.

149 Zacharias, C., Lemes, M. and Dal Pino, A. (1998). Combining genetic algorithm and simulated annealing: A molecular geometry optimization study. *Journal of Molecular Structure: THEOCHEM, 430*, 29-39.

150 Zegordi, S., Itoh, K. and Enkawa, T. (1995). Minimizing makespan for flow shop scheduling by combining simulated annealing with sequencing knowledge. *European Journal of Operational Research, 85*(3), 515-531.